FINAL REPORT

TITLE:

Advanced Workstations Accelerated by Embedded Massively Parallel Computer Modules for Image Processing Applications

SHORT TITLE:

Massively Parallel Computer Accelerator

ARPA ORDER NUMBER:

B-695 (Advanced Vision Systems I (AVIS I))

CONTRACTOR:

OC Inc.
5440 Cherokee Avenue
Alexandria, VA 22312

CONTRACT NO:

N00014-94-C-0195

REQUISITION NO:

c34f010---01 dated 25th May 1994

DATE OF CONTRACTS OF CONTRACT:

9-2-94 to 3-14-95

PRINCIPAL INVESTIGATOR:

Terence W. Barrett, Ph.D.

TELEPHONE NUMBER:

703-813-4160

TECHNICAL REPRESENTATIVE:

Barbara L. Yoon, Ph.D.
Program Manager
Microelectronics Technology Office
Advanced Research Projects Agency
3701 North Fairfax Drive
Arlington, VA 22203-1714

PROGRAM OFFICER:

Office of Naval Research
Ballston Tower One
800 North Quincy Street
Arlington, VA 22217-5660
Attn: ThomasS M. McKenna, ONR 342CN

Aspex
Microsystems Ltd

ARPA AVIS PROJECT

PHASE 1

FINAL REPORT

Document No: 325-006

Date of issue: 21st March 1995

Prepared by: Holger Kumm

Checked by: Prof R.M. Lea

Authorised by: Prof R.M. Lea

Aspex Microsystems Ltd.
Brunel University, Uxbridge
Middlesex, United Kingdom, UB8 3PH
Tel +44 (0)1895 274000 ext 2368, Telex 261173G
Facsimile +44 (0)1895 258728

DISTRIBUTION LIST

OCI Inc.:

Dr. Terence Barrett

Aspex Microsystems Ltd.:

Prof R.M. Lea
John Lancaster
Dr. Argy Krikelis
Dr. Ian Jalowiecki
Holger Kumm

Contents

1 Introduction

1.1 Background

This document presents the results of an investigation into the design options for cost-effective MPC (Massively Parallel Computer) accelerator modules for workstations, which was carried out as part of phase 1 of the ARPA AVIS program.

The main objective of the AVIS program is to develop, demonstrate and benchmark fully functional prototype hardware/software accelerator systems for both workstations or PCs and scalable parallel computers. The program is aiming at a substantial increase in computing performance for image-processing and other high-performance applications by integrating hardware, software, supporting technology and application domain expertise. Teaming between different centers of expertise is therefore strongly encouraged. Each developed system will include all software and hardware necessary to support application-specific demonstrations as well was application-specific and general purpose benchmarking.

The ARPA AVIS program is planned in two phases. Phase 1 was executed as a 6-months project. Phase 2 is planned as a multi-team, multi-project 36-months effort.

The project presented in this document aims at testing the feasibility and evaluating the Modular-MPC concept as an accelerator for image processing workstations. Carried out as part of the phase 1 effort, the project also aims at identifying the potentials of the Modular-MPC concept to contribute substantially in achieving the program goals of phase 2 of the ARPA AVIS project.

This document is divided into three main parts. In the first part (section 2) the results of a requirement analysis for computer accelerator technology in image processing is presented. Based on these results, the objectives for the remainder of the project are revisited and a strategy to meet the objectives is presented (section 3). The second part (sections 4, 5 and 6) is devoted a to case-study investigating and evaluating a specific implementation of the Modular-MPC in a HP 747i workstation [Hew93]. The final part of the document (section 7) presents conclusions and recommendations for phase 2. A detailed description of the Modular-MPC concept, Modular-MPC hardware architecture and Modular-MPC software architecture can be found in Appendix A.

The remainder of this section presents the statement of work as it was submitted with the proposal to phase 1.

1.2 Statement of work

The objective of this contribution to Phase I of the AVIS program is to study and define design options for cost-effective MPC (Massively Parallel Computer) accelerator modules for workstations which will greatly increase performance for image-related (e.g. medical

imaging) research and applications such as image reconstruction from tomographic data, image analysis and understanding, character recognition, video signal processing and 3-D graphics and simulation.

Building on existing Associative String Processor (ASP) technology, the design study will target generic fine-grain SIMD modules from which application-specific MIMSIMD MPC accelerators (satisfying performance and reliability requirements within size, weight, power and cost constraints) can be configured. Indeed, the study will consider the transition of proven hardware and software from parallel computing technology research to the engineering reality of cost-effective user environments (e.g. for medical imaging).

More specifically, the modules will be based on the emerging availability of a 256-processor VLSI ASP chip and experience gained in the experimental development of ASTRA (ASP System Test-bed for Research and Applications) hardware and software. In particular, the study will focus on MPC accelerators which can benefit from periodic technology upgrades, exploiting proven multichip and wafer-scale ASP module research and development, to achieve fourfold performance improvements due to upgrades in the number of processors from 16K (i.e. for a 128 x 128 pixel patch) through 64K (256 x 256) and 256K (512 x 512) to 1M (1K x 1K).

The program of work will include the following activities.

Requirements analysis

Study of image-related (e.g. medical imaging) application requirements (in close collaboration with users) and provision for them in popular workstations (e.g. from Hewlett-Packard, SUN and Silicon Graphics) including data sensors, format and rates, bus interface (e.g. S-bus, VME, PCI and proprietary) specifications and user/application software support. Selection of realistic (e.g. medical imaging) application benchmarks and definition of performance targets in conjunction with size, weight, power and cost constraints.

MPC accelerator hardware definition

Study of modular MIMSIMD configurations of fine-grain ASP modules and their integration within selected workstation hardware environments. Definition of a compatible set of generic ASP and data/instruction stream management boards from which application-specific MPC accelerators could be configured. The ASP board baseline will have 16K processors implemented with 4 4K-processor multichip modules, each incorporating 16 256-processor VLSI ASP chips. The targeted upgrade path is based on 16K-processor and 64K -processor plug-in replacements of these multichip modules.

MPC accelerator software definition

Study leading to the definition of ASP programming tools and ASP operating system for application-specific MPC accelerators and their transparent integration within workstation software environments (e.g. GUI). The ASP programming tools will be based on upgrades of the libraries, profiler and debugger pioneered on ASTRA and development of appropriate cross-compilers and linkers to enable C and C++ extensions.

MPC accelerator evaluation

Study leading to forecasts of the potential performance and efficiency of defined MPC accelerators for the selected application benchmarks and evaluation of their overall cost-effectiveness for workstation users.

Report

Documentation of the program results, describing the potential of ASP-based MPC accelerators and including MPC hardware and software module definitions, plans for their integration into selected workstation environments and a proposed program of work for Phase II of the AVIS program.

2 Requirement analysis

Image processing applications can be grouped into image analysis, video processing and image generation. Table 1 gives an indication of the variety of applications and, consequently, the different application requirements for image processing. Accelerators for image processing workstations have to be flexible enough to meet these requirements cost-effectively.

2.1 Image processing workstation accelerator requirements

Most image processing applications require the execution of a sequence of *task packages*, as indicated below for a typical computer vision application

object detection	image restoration *tasks* compensation for sensor non-uniformity (e.g. ageing and eventual failure), aberrations, blurring and lack of focus image enhancement *tasks* filtering for noise reduction, to emphasize particular signal features (e.g. edges), histogram equalisation and correlation to discriminate objects from background clutter
object analysis	object isolation *tasks* image segmentation and object labelling object quantification *tasks* extraction of object features and measurement of object properties (e.g. length, area, centre-of-mass coordinates) data reduction *tasks* iconic-to-symbolic conversion to list object properties
object recognition	object representation *tasks* description in terms of properties which are invariant to translation, rotation, scale and perspective hypothesis *tasks* association with data-base of known objects: model matching and object classification decision *tasks* interpretation (inference and prioritisation) and determination of appropriate response

Tasks control the evolution of specified sub-images and, typically, are application-specific in nature; the more complex usually comprising a hierarchy of *composition-tasks* and, at the lowest level *base-tasks* .

Although the terms *task* and *process* are widely abused and, therefore, effectively synonymous in computing folklore, it can be helpful to discriminate between them in order to distinguish two different types of image processing (and, as discussed in section 2.3, parallel computing) subprograms. *Task* is rarely misunderstood, but *process* can cause confusion. Hence, at the risk of upsetting certain pharisees of some operating system religions, *process* will be defined, for the purposes of this discussion, as follows.

Base-tasks are executed as a sequence of general-purpose *processes*, which control the navigation and evolution of specified data structures, as indicated below for typical *process* examples:

convolution	executed on	array	structures
search	executed on	table	structures
count	executed on	tree	structures
sort	executed on	graph	structures

This rather unusual interpretation of the term *process* should not be confused with the use of *process* in UNIX and some programming languages, which would be better described as

Image analysis

desktop/on-line computer vision
object detection, analysis and recognition
remote sensing
astronomy
geographic information systems: environmental monitoring
oil and gas exploration
high-energy particle-physics
medical (2D image) diagnosis
industrial assembly & inspection: robot vision
autonomous guided vehicles
military surveillance & tracking
security: fingerprint & face recognition
surveillance & tracking

Video processing

digital television (e.g. MPEG 2)
temporal/spatial compression, quantisation & vector-length coding
multimedia systems (education & entertainment)
video-on-demand
video/movie storage

Image generation

3D graphics (surface & volume rendering)
visualisation, simulation, modelling & virtual reality
animation
complex data analysis
education & entertainment
medical (3D image) diagnosis, therapy & surgery
scientific/engineering design
training

Table 1: Image processing workstation applications

task in this discussion. Nevertheless, this interpretation is consistent with the evolution of state (viz. data structure) more usually associated with *process* than *task* execution. Further such justification for this interpretation can be found in the discussion of section 2.4.

Processes are executed as a sequence of primitive *operations* (e.g. $+, -, x, and, or, <, =, >$

etc.) which control the evolution of pixels.

User and application requirements for image processing workstations can be summarised as follows:

Application-flexibility

Machine-versatility

As displayed in Table 1, image processing covers a broad range of different applications. Indeed, the disparate needs of varying imaging modalities, as evident in available sensors, interfaces and algorithms and, especially, in the need for efficient interaction between them, demand considerable hardware and software flexibility to match different application requirements.Moreover, as indicated above, far from constituting a single task, image processing applications require the end-to-end execution of a sequence of different *tasks*, *processes* and operations; thereby increasing the demand for versatility, in terms of configurability and programmability, in order to satisfy their different processing requirements.

User-acceptability

In order to achieve commercially success, image processing workstations must satisfy the requirements of the following two different types of user:

end-users	requiring support of existing image processing applications operating within a familiar graphical user environment
application developers	requiring software development support within a familiar (e.g. C or C++) programming environment

Performance-scalability

Naturally, a workstation family should cover the full range of performance requirements, which, for image processing applications, are dominated by frame-size and frame-rate. In particular, real-time image processing imposes arduous performance requirements.

Figure 1 displays performance and data rate requirements for a range of frame-sizes, frame-rates and algorithm requirements from 100 (e.g. for a 3×3 convolution) up to 100,000 (e.g. for a sequence of 2D/3D processing tasks) operations (i.e. 8 - 16 bit integer additions) per pixel. Indeed, the graph demonstrates that, for typical workstation frame-sizes, performance-scalability in the range from 10 Giga-OPS to 1 Tera-OPS is required for near real-time execution.

Cost-effectiveness

Size, weight and power constraints

In order to achieve widespread acceptance among users, image processing workstation accelerators should comply with reasonable size and weight constraints, of the order of

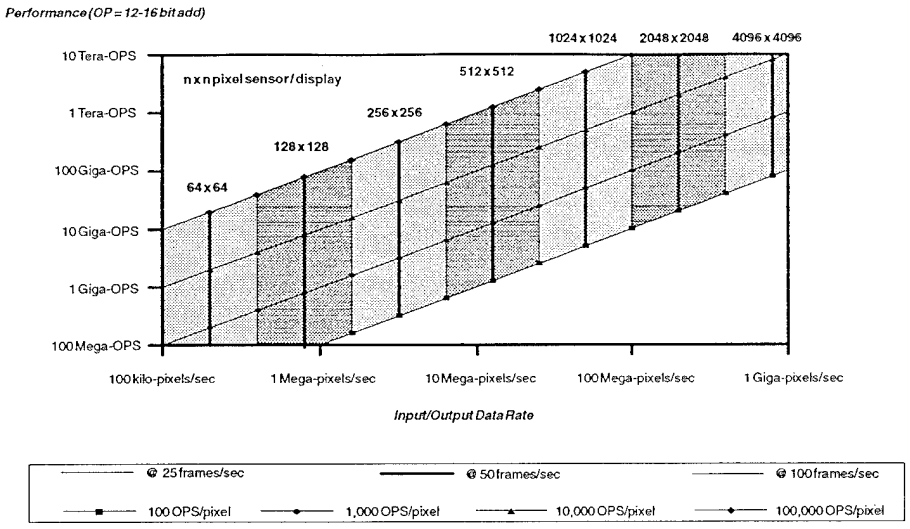

Figure 1: Image processing performance and data requirements graph

$1 ft^3$ and $10 lbs$, and restrict power to significantly less than 1kW; thereby requiring an implementation technology capable of delivering the figures-of-merit shown in Table 2.

Cost limitations

Market surveys suggest that, in order to gain a significant sales volume, entry-level image processing workstation accelerators, capable of 10 - 100 times performance improvement should cost less than $20k. Similarly, an image processing workstation equipped with the most sophisticated accelerator, capable of 100 - 1,000 times performance improvement, should cost no more than $100k, thereby requiring an implementation technology capable of delivering the figure-of-merit shown in Table 2.

Operational-efficiency

Optimal cost-effectiveness requires machines to operate at maximal computing efficiency in order to maintain performance levels close to their potential. Consequently, maintaining high operational-efficiency over the image processing range is an essential requirement for cost-effective workstation accelerators.

Future-proofing

Adoption of any new implementation technology, for a computing product, involves considerable investment; not only in terms of hardware and software procurement costs but also in development costs, when climbing the learning curve associated with applying the new technology efficiently, and retraining of engineering and marketing staff. Moreover, inevitable progress in microelectronics technology, ensures a limited lifetime for new computer components. Thus, return on investment, before a new implementation technology becomes outdated, is of keen concern to its potential users. Accordingly, protection of user interests, in terms of extending product lifetimes with mid-life upgrades is an essential requirement for image processing workstation acceler-

ators. Indeed, cost-effectiveness in this market sector necessitates such future-proofing in addition to cost-performance potential.

	Acceptable targets			Figures-of-merit	
size	$<$ 1	ft^3	$>$ 1	$Tera-OPS/ft^3$	
weight	$<$ 10	lbs	$>$ 100	$Giga-OPS/lb$	
power	$<$ 1	kW	$>$ 1	$Giga-OPS/W$	
cost	$<$ 0.1	$\$M$	$>$ 10	$Mega-OPS/\$$	

Table 2: Target implementation technology figures-of-merit for image processing workstation accelerators

2.2 Current workstations

Popular (e.g. Sun, HP, Silicon Graphics) workstations support image processing, providing commendable versatility and setting the standard for *user-acceptability* and *future-proofing*, as indicated in Table 3. They also set standards for *size, weight, power* and *cost*, but, currently offering around 100 MIPS, they fall short of *performance-scalability* requirements (i.e. 10 Giga-OPS - 1 Tera-OPS) by between 2 and 4 orders-of-magnitude! Accordingly, they may require minutes or even hours to perform the more complex image processing applications, which, lack realism and fail hopelessly to satisfy "real-time" constraints.

Application-flexibility	
Machine-versatility	+
User-acceptability	0
Performance-scalability	- - -
Cost-effectiveness	
Size, weight & power	- - -
Cost	- -
Operational-efficiency	-
Future-proofing	0

Table 3: Relative satisfaction of image processing requirements by current workstations

With reference to Table 2, current workstations fall short of the required figures-of-merit by between 2 and 4 orders-of-magnitude in terms of *size* and *weight*, between 3 and 4 orders-of-magnitude in power and between 2 and 3 orders-of-magnitude in *cost* . Since long-term forecasts for sequential microprocessor fabrication technology conservatively approach only 1 (and the most aggressively only 2) orders-of-magnitude improvement in such figures-of-merit, it is clear that the required step-functions will not be achieved by this route. Accordingly, interest is turning again to parallel processing accelerators for image processing workstations.

2.3 Natural-parallelism in image processing

Image processing provides excellent opportunities to exploit natural-parallelism.

In terms of *tasks* and *processes*, as defined in section 2.1, multiple *task* execution would exploit control-level natural-parallelism and *process* execution would exploit data-level natural-parallelism, since the multiple nodes of the processed data structure could evolve (and the links between them could be traversed) concurrently.

Note that the distinction between *tasks* and *processes*, introduced in section 2.1, is also helpful to distinguish between these two fundamentally different types of natural-parallelism. Indeed, the interpretation of *process* in this discussion is also consistent with the common use of the term *multiprocessing* (i.e. concurrent *process* execution) for multiple *task* execution.

Opportunities for massive data-level natural-parallelism are evident in operations to be executed on large subsets of the pixels constituting an image, since these pixel values could, theoretically, be updated simultaneously. In contrast, only modest control-level natural-parallelism is evident in the much smaller number of pixels which it might make sense to process independently.

Whereas data-level natural-parallelism is limited only by the number of pixels in the image, control-level natural-parallelism is limited by the complexity of the mathematical algorithm controlling multiple operation execution. For example, continuing the computer vision example given in section 2.1, limited control-level natural-parallelism could be derived from continuous input data streams; by pipelining the execution of the detection, analysis and recognition *task packages* for consecutive image frames. Similarly, within an image frame, it is very unlikely that applications would require individual pixels to be processed differently. Indeed, a group of pixels is required to represent each object in an image and, although different classes of object could be discriminated meaningfully and processed differently, the number of different object classes would be very much less than the total number of pixels.

2.4 Current Massively Parallel Computers (MPCs)

The last decade has seen many attempts to apply SIMD (Single Instruction control of Multiple Data streams) and MIMD (Multiple Instruction control of Multiple Data streams) MPCs to image processing.

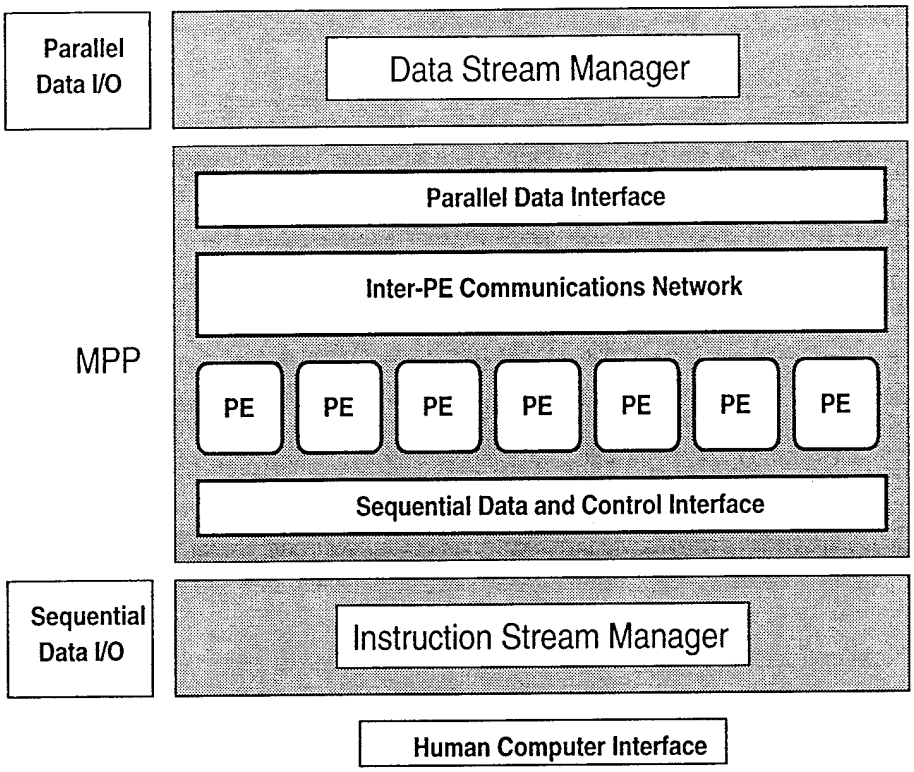

Figure 2: Massively Parallel Computer (MPC) schematic

As shown schematically in Figure 2, MPCs incorporates 3 main functional units, a Data Stream Manager (DSM), a Massively Parallel Processor (MPP) and an Instruction Stream Manager (ISM). The MPP comprises an ensemble of processing-elements (PEs), an Inter-PE communication network, a parallel data interface with the DSM and a sequential data and control interface with the ISM.

Despite their promising potential, current MPCs have failed to achieve the *application-flexibility* and *cost-effectiveness* required for image processing applications, as indicated in Table 4.

Although general-purpose MPCs (e.g. Paragon, Maspar, DAP, Transputer etc.) have demonstrated *performance-scalability* (but rarely up satisfying real-time requirements), such machines not only fall short of the *size, weight, power* and *cost* figures-of-merit by between 2 and 3 orders-of-magnitude but they also fail to satisfy *versatility* and *efficiency* (see subsection 2.1) requirements.

Structural mismatch between the varying natural-parallelism across different applications and the rather rigid applied-parallelism of most current MPCs has led to "force-fitting" of parallel algorithms on unwieldy MPC architectures and consequent inefficiency in their execution.

Application-flexibility	
Machine-versatility	- -
User-acceptability	- - -
Performance-scalability	-
Cost-effectiveness	
Size, weight & power	- -
Cost	- -
Operational-efficiency	- -
Future-proofing	- - -

Table 4: Relative satisfaction of image processing requirements by current MPCs

Moreover, poor parallel programming environments based on machine-oriented software interfaces, forcing reluctant programmers to wrestle with often bizarre architectural details, detract further from efficient MPC deployment with woefully inadequate *user-acceptability.*

Such experiences have led some users to develop application-specific MPCs, but lack of appropriate experience and resources, higher than anticipated costs, inevitable delays and the frustration of being sidetracked away from the work which originally motivated the image processing development, combine to deter all but the most steadfast teams. However, even these groups are likely to succumb to the daunting tasks of debugging and maintaining the hardware and software components of their creations.

Worse still, the *future-proofing* of current MPCs remains a cause of major concern for their commercial exploitation.

Nevertheless, despite this disappointing outcome for first-generation MPCs, progress up the architectural learning-curve and improvements in implementation technology herald second-generation MPCs which should come much closer to realising their potential.

2.4.1 Parallel computing efficiency

Despite its simplicity, the analysis presented below offers helpful insight into the factors detracting from MPC performance and, thereby, helps to focus attention on the more important design decisions for MPC algorithm and architecture developments.

With regard to natural parallelism,

S_n = potential relative speed-up achieved by ideal MPC due to natural-parallelism

$$S_n = \frac{T_s}{T_x} = \frac{n}{n \cdot T_{xav}} = \frac{1}{\sum_{i=1}^{n} P_n(i) \cdot \frac{1}{i}}$$

where
- Ts = time to execute n operations on a sequential computer
- T_x = time to execute n operations on an ideal MPC
- T_{xav} = average execution time per operation
- $P_n(i)$ = probability of natural-parallelism allowing i operations to be executed simultaneously on an ideal MPC

The expression for S_n and, especially, comparison of its value with n, provides a measure of the limitations of natural-parallelism (or the degree to which an application can be parallelised) and, therefore, its suitability for parallel processing. Moreover, the expression demonstrates the dominating influence on S_n, of

$P_n(1)$: inevitable sequential processing

$P_n(i)$: (for low values of i) low natural parallelism

The expression indicates clearly that MPCs should not be considered as a cost-effective solutions for general computation. Indeed, MPC solutions should only be sought for those applications with minimal values of $P_n(1)$ and $P_n(i)$ for low i. Fortunately, many image processing *tasks* fall into this category, due mainly to their inherently massive data-level parallelism.

With regard to applied parallelism,

S_r = relative speed-up achieved by real MPC due to its applied-parallelism

$$S_r = \frac{T_s}{T_p} = \frac{n}{T_{oh} + T_x} = \frac{T_x}{T_{oh} + T_x} \cdot S_n$$

$$= \left[1 - \frac{T_{oh}}{T_{oh} + T_x}\right] \cdot S_n = \left[1 - \sum_{i=0}^{N} P_{oh}(i)\right] \cdot S_n$$

where
- T_p = time to execute operations on a real MPC
- T_{oh} = time penalty due to overheads on a real MPC (i.e. extra time beyond that required for the execution of natural-parallelism)
- $P_{oh}(i)$ = probability of applied-parallelism overheads occurring in mode i

The expression for S_r and, especially, comparison of its value with S_n and N, provides measures of the exploitation of natural-parallelism (i.e. potential speed-up) and the utilisation of available PEs respectively achieved by an MPC. Moreover, the expression demonstrates the deleterious influence on S_r, of

$P_{oh}(0)$: housekeeping (e.g. input-output data transfers, communication between PEs, control set-up, status monitoring etc.)

$P_{oh}(1)$: unnatural sequential processing

$P_{oh}(i)$: (for $2 \leq i \leq N$) applied/natural-parallelism mismatch

enforced by the limitations of MPC architecture or MPC algorithm designs. Unfortunately for current MPCs, values for these probabilities have been allowed to approach unity, thereby contributing to their disappointing outcome. Indeed, $P_{oh}(i)$ must be ruthlessly minimised before users can benefit from potential MPC performance.

2.4.2 Image patching

Ideally, an image processing MPC would provide an individual PE, of appropriate computational power, for each pixel in the image. However, high MPC procurement costs ensure that this ideal state is very rarely approached in reality. Thus, in practice, MPC accelerator performance will be dominated by image patching.

Since the number of pixels in an image frame ($p \times q$) is typically much larger than the number of available processors (N), algorithms must be repeated for each of p.q/N image patches. For example, for a 4,096-PE MPP, a $1,024 \times 1,024$ pixel image would require at least 256 repetitions. Such patch repetitions would be greatly increased for 3D image processing. Moreover, for the same MPP, a $256 \times 256 \times 256$ voxel image would require at least 4,096 repetitions!

Such patching problems contribute greatly to $P_{oh}(i)$, as defined in subsection 2.3. Indeed, the overheads inevitably associated with each patch are repeated, all algorithm repetitions constitute additional overheads and patching itself induces further overheads. Moreover, for smaller patch sizes, patch overlapping and inter-patch data transfers can force the number of patch repetitions to greatly exceed $p \cdot q/N$.

2.4.3 Balancing DSM-MPP data transfer bandwidth with MPP processing power

The *operational efficiency* of an MPC, for image processing applications, is particularly sensitive to data transfer delays associated with the DSM-MPP interface (see Figure 2).

Indeed, very careful pipelining and overlapping of data transfer and patch processing are required, in order to minimise $P_{oh}(0)$, as defined in subsection 2.4.1.

The larger the MPP (i.e. the higher the number of PEs), the lower the number of patch repetitions and, hence, the lower the patching related overheads contributing to $P_{oh}(i)$. However, patch data transfer times increase with MPP size. Hence, image buffering is required within the DSM, which must support the external frame- rate (e.g. 30fps for real-time) and, internally, must support a much higher patch-rate; because the MPP must complete all patch processing repetitions within a single frame-time. Consequently, multiple DSM-MPP data channels are required to increase patch-rate and, thereby, avoid increase in the value of $P_{oh}(0)$.

Thus, the *operational efficiency* of an MPC can be "tuned", by varying the number of DSM-MPP data channels and the number of PEs allocated per channel until the "balance-point" between "I/O-bound" and "process-bound" computational states is achieved. Increasing the number of channels can pull the MPC out of the former state and increasing the number of PEs can pull the MPC out of the latter state. However, as indicated in Figure 3, when "process-bound" (or, conversely, "I/O-bound"), further increases in the number of PEs (or, conversely, channels) could have no beneficial effect on performance and, indeed, would detract from *cost-effectiveness* .

Two constraints set the limits of this "tuning" range; the upper being image input-output time (typically 33ms) and the lower being patch processing time (e.g. $10\mu s - 1ms$, depending on algorithm complexity).

Note that a range of "balance-points" exists (i.e. the "ridge of peaks" shown in Figure 3), corresponding to the minimum numbers of data channels and PEs required to achieve a target performance level. Consequently, optimising *cost-effectiveness* is a matter of climbing the " ridge of peaks" until the critical "balance-point", at which the MPP can keep up with external frame-rate, or the highest "peak", representing the *cost-effective* maximum performance of the MPC, is reached.

Unfortunately, the diversity of image processing requirements (see Table 1) is such that the "critical balance-point" is different for each application. Indeed, image processing application studies reveal that "critical balance-points" lie in the range of 1 - 64 DSM-MPP data channels and 256 - 16,384 PEs per channel.

Clearly, although MPC architectures, comprising a fixed configuration of channels and a fixed PE allocation per channel, may achieve the "critical balance-point" for certain specific applications, they will lack *cost- effectiveness* for the majority of the range of image processing applications.

Conversely, architectural configurability, to achieve the "critical balance-point" for each image processing application, becomes an essential requirement for MPC *cost-effectiveness* .

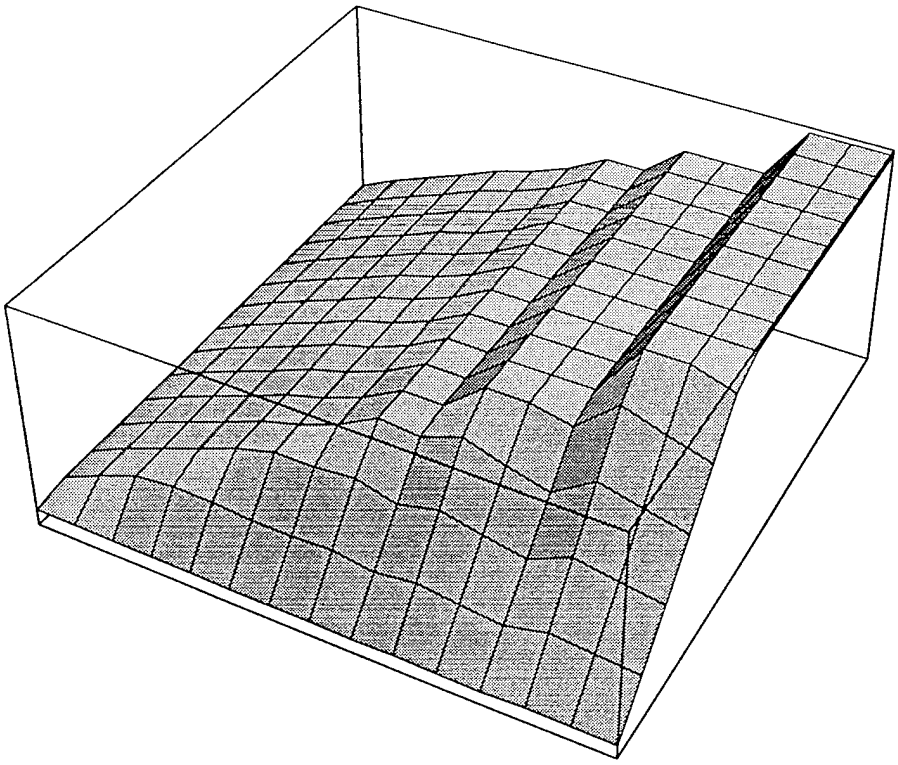

Figure 3: 3D graph: Perf vs # channels vs # PEs

3 Research program

3.1 Objectives

The project has three main objectives

1. Study, define design options and investigate the feasibility of the Modular-MPC concept
2. Estimate the cost-effectiveness of the Modular-MPC concept
3. Make Recommendations for phase 2 of the ARPA AVIS project

3.2 Strategy

In view of the severe shortcomings of current MPC in terms of *application flexibility* and *cost-effectiveness* (see Table 4 in section 2.4), the investigation into cost-effective accelerators for image processing workstations was based on the Modular-MPC. Emerging from recent

research at Brunel University, the proposed Modular-MPC architecture is designed to address the issues discussed in section 2.1. The Modular-MPC hardware architecture is introduced in Appendix A.2, its software architecture is presented in Appendix A.3. The Modular-MPC can be implemented as a vendor-neutral add-on or embedded image processing workstation.

In order to test the feasibility of the Modular-MPC as an accelerator for image-processing workstations, the template (Table 3) introduced in section 2.1, which was used to assess other MPCs (section 2.4) was applied to the Modular-MPC. Thus, by subjecting the Modular-MPC to the same rigorous criteria which have been used in the requirement analysis (section 2), it was possible to check whether the Modular-MPC can meet the requirements where other MPCs have failed.

Therefore, the pragmatic way to proceed was with a case-study. Only in this way can the existing Modular-MPC architecture be subjected to real problems and its theoretical concepts be tested in a realistic environment.

Although the Modular-MPC architecture has vendor-neutral implementations and can be either attached or embedded, it was tested for a particular embedded implementation. Thus, tougher requirements in terms of size, power and cooling were imposed. These specific, and therefore realistic, constraints were defined by the host-workstation. In fact, via the case-study, it was possible to test which configurations of the functional blocks of the proposed Modular-MPC architecture can be implemented within the constraints of the host workstation. Furthermore, embedding the Modular-MPC in an existing workstation provides the opportunity to open a quicker route to market.

The Hewlett Packard HP 747i workstation [Hew93] was chosen as a vehicle for the case-study. It is part of the HP 9000 series which targets the high-end image processing market. User surveys of application users revealed preferences for Hewlett Packard, Silicon Graphics and Sun workstations. However, in military and medical areas where accelerators for image-processing workstations could have an impact, a clear preference for Hewlett Packard workstations can be observed. On the other hand the HP 747i is very suitable as a host-workstation for embedded accelerators, since, as an industrial workstation which can be rack-mounted, it specifically provides for user add-on hardware. In fact, with the HP workstation product range being based on a modular concept, the Modular-MPC could be seamlessly integrated as an additional module. Therefore, with its combination of existing customer-base and modular flexibility, the HP 747i workstation could provide a promising route to market.

In order to test the feasibility of the Modular-MPC, the aim was to provide a Modular-MPC design study which would be detailed enough to be picked up at a later stage for direct implementation. Based on this design study real data could be collected which could consequently be used to evaluate the Modular-MPC concept in terms of the image processing application requirements introduced in section 2.1 (see Table 3).

In order to provide realistic application requirements, application exemplars which provide a cover of image processing and could be used to benchmark the Modular-MPC were selected (see section 3.2.1).

The investigation into the feasibility and cost-effectiveness of the Modular-MPC was carried out in three phases:

Phase 1: Study design options and define a set of boards for a particular implementation of the proposed Modular-MPC architecture to fit the constraints of the HP 747i workstation (see section 3.2.2)

Phase 2: Evaluate the application flexibility and cost-effectiveness of the Modular-MPC for each application exemplar (see section 3.2.3)

Phase 3: compare the Modular-MPC results with published data for other MPCs (see 3.2.4)

3.2.1 Selection of exemplars

Two categories of potential users for image processing workstation accelerators can be observed. The majority of users will normally only be interested in the overall performance of their end-to-end application. However, other users, especially those using application packages (e.g. Analayze [RH90], IDL [Res94], Photoshop [GS95]), are also interested in the performance of particular image-processing tasks. Therefore, in order to provide a sufficient cover of image processing tasks and to be able to benchmark the Modular-MPC, two types of exemplars were selected:

- **End-to-end application exemplars**

  Only the study of end-to-end application exemplars gives relevant information to users on the performance of integrated solutions and the *machine versatility*. This has been pointed out repeatedly and considerable effort is put into the definition of benchmark suites which consist of end-to-end applications (see for example [WRHR91]). In order to evaluate the Modular-MPC as an accelerator for image processing workstations, exemplars of the three main areas of image processing (see Table 1), image analysis, video signal processing and image generation, were selected.

  - **Image analysis**

    Near Earth Object (NEO) Detection
    DARPA Image Understanding Benchmark II

  - **Video signal processing**

    data compression (including motion compensation)

  - **Image generation**

    geometric transformations followed by
    surface rendering

volume rendering

These exemplars are introduced in more detail in section 5.2.

- **Typical image processing task**

  In order to complement the end-to-end application exemplars and to provide a broad cover of image processing, several exemplars of typical image processing tasks, which would normally be provided as library modules within image processing application packages were selected:

  - convolution
  - median filter
  - sobel filter
  - thresholding
  - histogramming
  - normalise
  - Fast Fourier Transform (FFT)

3.2.2 Study design options and define a set of boards for the Modular-MPC

In this phase of the case-study, a design study within the restrictions of the HP 747i workstation was carried out. In order to ensure that this design study is carried out in the most realistic environment possible two further restrictions were imposed:

- only published components were used for the board design
- only size, power and performance values published in data sheets were used

The design study was carried out in an iterative manner. Modular-MPC boards and their assembly were proposed and subjected to an electrical analysis, a thermal analysis and a cost analysis. Based on the results of these analyses, designs were reconsidered and optimised until an acceptable solution was found.

3.2.3 Evaluation of application flexibility and cost-effectiveness

In the evaluation phase of the case-study, each exemplar was subjected to the Modular-MPC methodology and evaluated:

- Analysis of the natural parallelism (see section 2.3) and development of algorithms and library modules on the ASP simulator [Asp90].

- Demonstration of the application exemplar on the ASP System Test-bed for Research and Applications (ASTRA) [Asp93], [Asp94a]

  This demonstration on a real system served two purposes. A functional verification could be carried out for each application exemplar, and performance data could be gathered which provided the basis for a realistic performance forecast for the Modular-MPC.

- Synthesis of a Modular-MPC configuration to match the natural parallelism of the application

  The parallel computing efficiency (see section 2.3) of the derived configuration was estimated and, in an iterative manner, optimised.

- Analysis of the *cost-effectiveness*

  The performance of the particular Modular-MPC configuration, which had been synthesised for an application exemplar was forecasted based on

  - the performance data gathered during the application demonstration on ASTRA
  - the performance predictions for the Modular-MPC, based on published data sheets, which emerged from the design study

  The cost forecast was based on

  - the hardware requirements of the derived Modular-MPC configuration
  - the cost predictions for the Modular-MPC, based on price quotes for 100+ components, gained in the design study

3.2.4 Comparison with other MPCs

The aim of this phase of the case-study was to put the Modular-MPC performance into perspective. To this end, MPCs which target the image processing application area were selected. Available data on performance, size, power and cost was gathered and a comparison with the results of the Modular-MPC was attempted.

4 Case study phase 1: Modular-MPC Hardware design study

The objective of this phase of the Modular-MPC case study was to study design options and define a set of boards for a particular implementation of the Modular-MPC architecture (see Appendix A.2) within the restrictions of the HP 747i workstation (see [Hew93] and section 3.2). It should be pointed out, once again, that this design study did *not* aim to

define *the* implementation of the Modular-MPC architecture presented in Appendix A, but rather to test the feasibility of the different theoretical concepts introduced in Appendix A.2 in a realistic environment. In fact, the Modular-MPC architecture can have many different implementations and any performance, cost, etc. parameters derived in this section only apply for the specific design study which is presented.

Although the Modular-MPC architecture introduced in the previous section is unrestricted in its scalability and modular adaptability, any implementation of this architecture will impose particular restrictions. Therefore, a main aim of the design study was to investigate which levels of scalability and modularity could be sustained for a specific implementation in a realistic environment.

As shown in Appendix A.2, the Modular-MPC architecture supports MIMSIMD configurations of Task Execution Units (see Figure 24). However, such configurations were not considered for the design-study presented in this section. It can be shown, that as long as it is not possible to harness the data-level parallelism to its full extent, it is not cost-effective to attempt to make use of the control-level parallelism through a MIMSIMD configuration. In fact, for applications which require overlapping of patches (see section 2.4.2), reducing the degree of exploited data parallelism in favour of control-level parallelism can result in a considerable performance reduction due to the non-linear relation of the patch-size and the number of patches (see section 2.4.2).

The analysis of the application exemplars showed, that all exemplars can be implemented more cost-effectively on a single Task Execution Unit.

As a first step of the design study, a board partition for the Modular-MPC architecture was derived. The aim was to define a partition which is modular enough to implement each Modular-MPC configuration with a minimum hardware effort. Obviously, the board partition is restricted by the available space. In the case of the HP747i workstation, boards with the 6U form-factor (i.e. $233.35mm \times 160mm$) have to be used.

Having studied the requirements of application exemplars (see section 3.2.1), the board partition and modular options shown in Figure 4 emerged to be essential in order to provide a cost-effective cover for all exemplars.

A Modular-MPC configuration can include

- 1 ISM/DSM mother-board
- up to 4 DSM-SDS daughter-boards
- up to 2 DSM-PDR daughter-boards (very rarely up to 4, see section 4.3)
- 1 to 4 MPP daughter-boards

The main features and the scalability of this particular implementation of the Modular-MPC architecture are listed in Table 5.

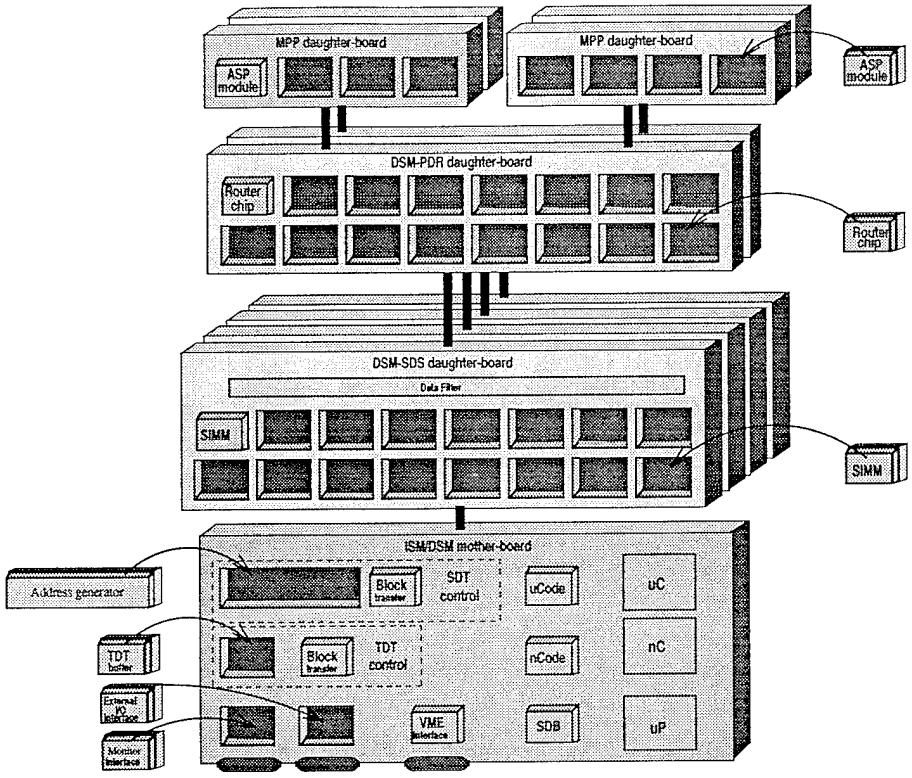

Figure 4: Modular-MPC configuration options

Thus, the basic configuration for an entry-level Modular-MPC includes 2 boards (ISM/DSM mother-board, MPP daughter-board)

- 256 - 4k - 16k PEs
- 1-stage or 2-stage parallel data I/O pipeline (see Figure 28)
- up to 4 parallel I/O data channels for SDT
- up to 1M 32-bit Secondary Data Store

This configuration provides an Instruction Stream Manager (ISM, see Appendix A.2.4) with the parallel data I/O control in its minimal configuration (see Appendix A.2.4.4), a Massively Parallel Processor (MPP, see Appendix A.2.2) and 1-stage or 2-stage Data Stream Manager (DSM, see Appendix A.2.3). The DSM can be configured as a 1-stage or 2-stage parallel data I/O pipeline by configuring the ISM/DSM mother-board with the appropriate memory (see section 4.4).

On the other end of the spectrum, the maximum configuration for this particular implementation is configured with 11 boards. This configuration implements a 3-stage DSM (see

- 256 - 16k - 64k PEs (i.e. up to 109.23 GOPS (op = 12-bit add)
- 1- to 3-stage parallel data I/O pipeline (see Figure 28)
- 1-64 low-cost parallel data I/O channels for high bandwidth SDT with a transfer rate of 40 Mitems (32-bit) per second
- 32k to 16M 32-bit Secondary Data Store (see Figure 29), which provides storage for example for 16 $1k \times 1k \times$ 32-bit 2D images or 2 $256 \times 256 \times 256 \times$ 16-bit 3D images.
- 1-4 low-cost parallel data I/O channels for TDT (see Figure 30) with a transfer rate of 40 Mitems per second.
- 32k to 1M 32-bit Tertiary Data Queue (see Figure 30) which provides buffering for example for 1 $1k \times 1k \times$ 32-bit 2D image
- optional Parallel Data Router for 2-64 data channels (see Figure 29)
- optional external I/O interfaces, which support different high-speed data transfer protocols with transfer-rates of up to 800 MBits per second, and monitor interface

Table 5: Features of the Modular-MPC implementation meeting the restrictions of the HP 747i workstation

Appendix A.2.3) which includes a shared SDS (see Appendix A.2.3.1). The parallel data I/O control in the ISM (see Appendix A.2.4.4) therefore has to comprise all modules shown in Figure 33.

The HP 747i workstation offers 6 VME extension slots which provide the space for the Modular-MPC implementation. However, as indicated above (see Figure 4), the maximum configuration of the Modular-MPC consists of 11 boards. Also, the data transfer rates of the VME interfaces do not match the transfer rates required in the DSM. The Modular-MPC was therefore implemented as a stack of boards, which provides much denser packing of boards than it would be possible with a standard VME spacing. Figure 5 shows a Modular-MPC assembly example.

In this particular example, the Modular-MPC is configured with one ISM/DSM mother-board, one DSM-SDS daughter-board and four MPP daughter-boards.

Each stack starts with an ISM/DSM mother-board, on which daughter-boards can be stacked. The boards are connected to each other via CIN::APSE standard stacking connectors (see [Chi91]). In order to keep the boards as generic as possible, it is desirable that the com-

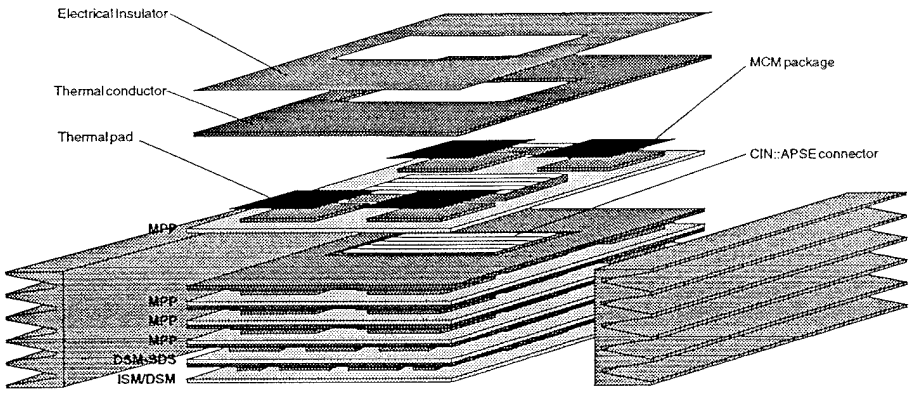

Figure 5: Modular-MPC SIMD assembly example 1

ponents on a board (e.g. MPP daughter-board) connect to the same pads of the stacking connector, independent of the position of the board in the stack. Figure 6 shows a strategy which achieves this (note that the ISM/DSM mother-board is not shown in this diagram).

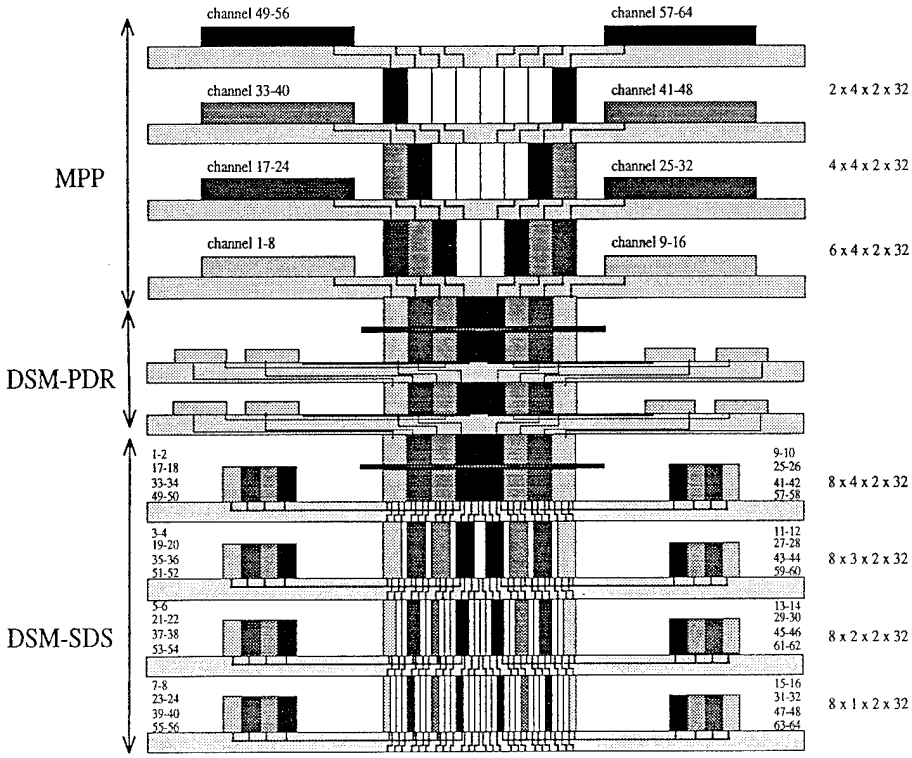

Figure 6: Modular-MPC board stacking strategy

Signals are routed in a tree-like fashion through the stack. On each level of the stack all signals which are assigned to components on the same boards are routed one level outwards (on MPP daughter-boards) or inwards (on DSM-SDS daughter-boards). Thus, the compo-

nents on a board can always connect to the outermost layer of connection in the stacking connector and receive their assigned signals.

The Modular-MPC stack, together with its cooling fins is then mounted in a cage, which plugs into the workstation as a single module. Thus, the Modular-MPC SIMD assembly is connected to only one of the host's VME expansion slots via the ISM/DSM mother-board.

Figure 7 shows the configuration example introduced in Figure 5 as an assembled stack together with its relevant dimensions. A standard 6U format is used for all boards in the stack.

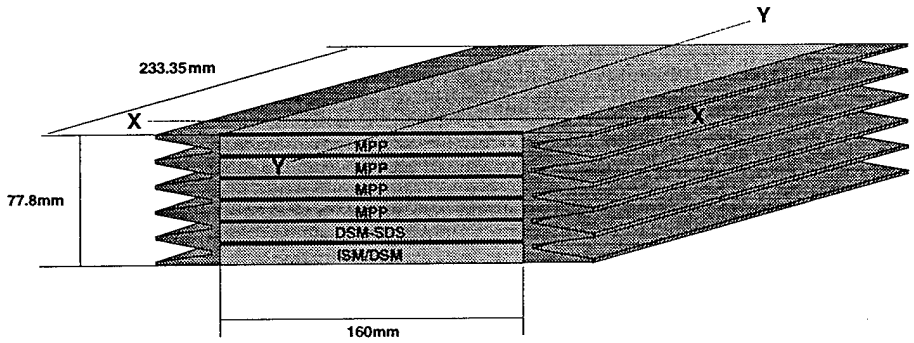

Figure 7: Modular-MPC SIMD assembly example

The cross sections indicated in Figure 7 are depicted in Figures 9 and 8.

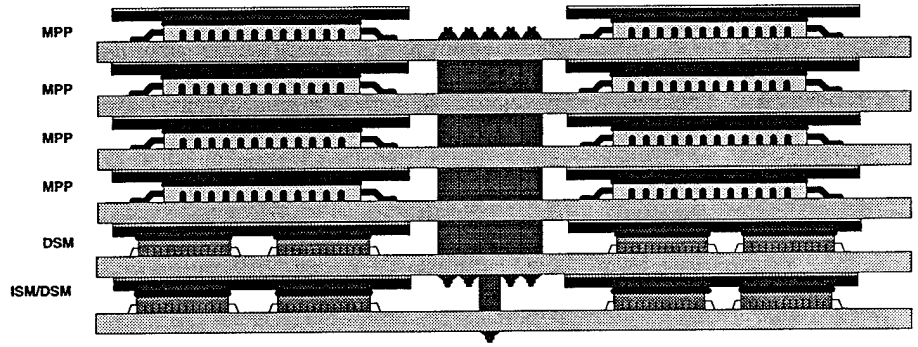

Figure 8: Modular-MPC SIMD assembly example: Y-Y cross section

The Y-Y cross section (Figure 8) shows the stacking of the boards in more detail. Each stacking connector joins pads on top of the lower board with pads on the bottom of the upper board. The boards are clamped together in order to keep the stacking connectors in place and to create good connections between the boards

The X-X cross section in Figure 9 demonstrates the cooling strategy used for the particular implementation example, which has already been indicated in Figure 5. The components on each board are connected to a thermal conduction plate via thermal pads. Thus, the heat is conducted from the components via the thermal pads and the conduction plate to the edges of the boards where it is dissipated on fins via forced-air cooling.

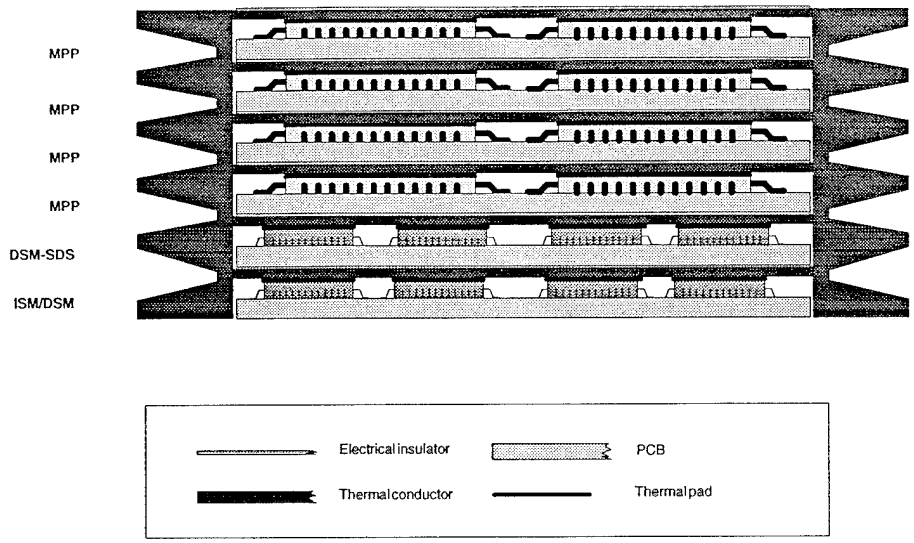

Figure 9: Modular-MPC SIMD assembly example: X-X cross section

Figure 10 shows the assembly example in a racking enclosure. A top view demonstrates how the air-flow is forced along the cooling fins of the stack. Furthermore the two VME connectors which provide the only connection between the Modular-MPC SIMD assembly and the HP 747i host workstation are visible.

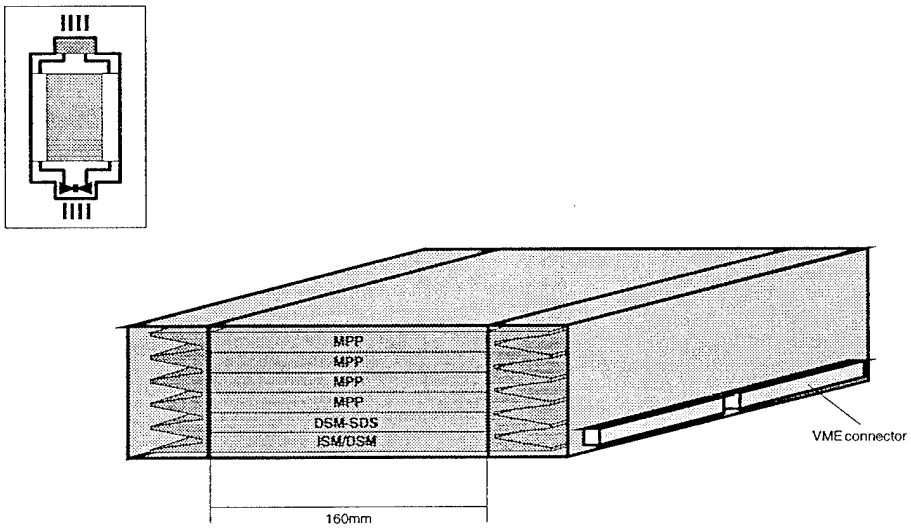

Figure 10: Modular-MPC SIMD assembly in racking enclosure

A first analysis of this cooling strategy demonstrated that the required air-flow along the fins is well below the air-flow generated by the fans inside the HP 747i workstation.

The cooling strategy introduced above is only required for Modular-MPC SIMD assemblies which include a large number of boards. For smaller assemblies, with relaxed size constrains, an alternative approach to cooling demonstrated in Figure 11 can be taken and the cooling strategy discussed above does not have to be used for the whole stack.

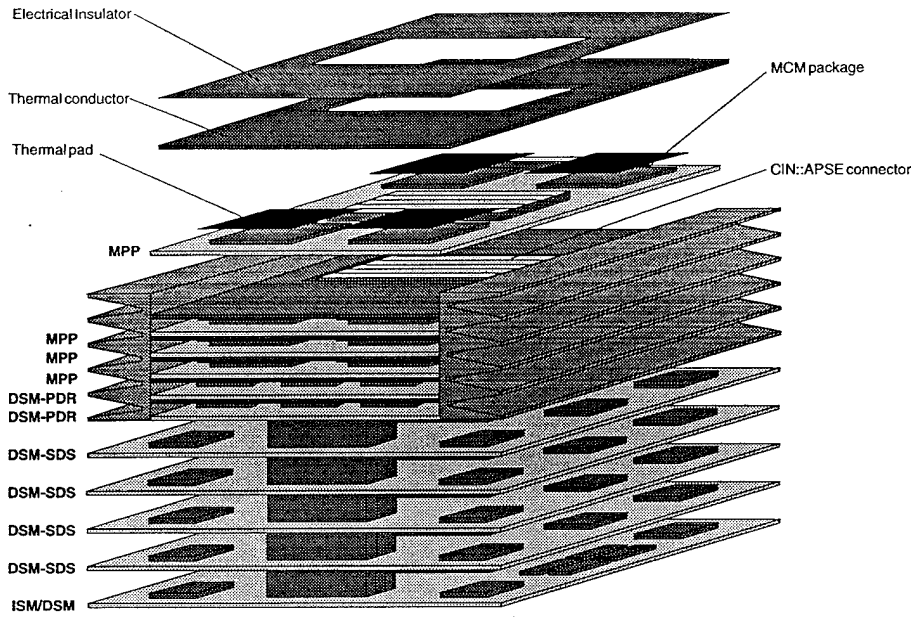

Figure 11: Modular-MPC SIMD assembly example 2

Boards with a simple profile (DSM-PDR and the MPP daughter-boards) are cooled as described above, through conduction and forced air-cooling along fins. All other boards which expose a more complex profile are cooled directly via forced-air cooling. This would have the advantage of simplifying the stack assembly process. However, the direct air-cooling approach requires bigger spacing between the boards leading to an increased stack height.

In the remainder of this section the different boards are introduced in more detail. All boards have been subjected to an electrical analysis. Quoted data transfer rates are based on simulations which incorporated performance parameters for the used components based on their data sheets.

4.1 MPP daughter-board

Figure 12 shows the MPP daughter-board. It implements the MPP introduced in Appendix A.2.2. Each MPP daughter-board includes 1 to 4 ASP modules, where each module is connected to 4 32-bit parallel data channels. Each parallel data channel, as well as the sequential data channel is driven separately in order to sustain the parallel data transfer rate of 40 Mitems/sec (32-bit items). The same applies for the (common) sequential data channel and the control bus, which are driven separately on each MPP daughter-board.

Figure 13 shows an ASP module implemented as a Multi Chip Module (MCM).

The Modular-MPC MCM holds 1 to 4 HDIs (High Density Interconnects) with 1 to 4 ASP chips. Each chip implements 256 PEs, thus, in its full (and most cost-effective) configuration the Modular-MPC MCM implements 4k PEs. Furthermore, each input channel of the Modular-MPC MCM is connected to 256-1k PEs.

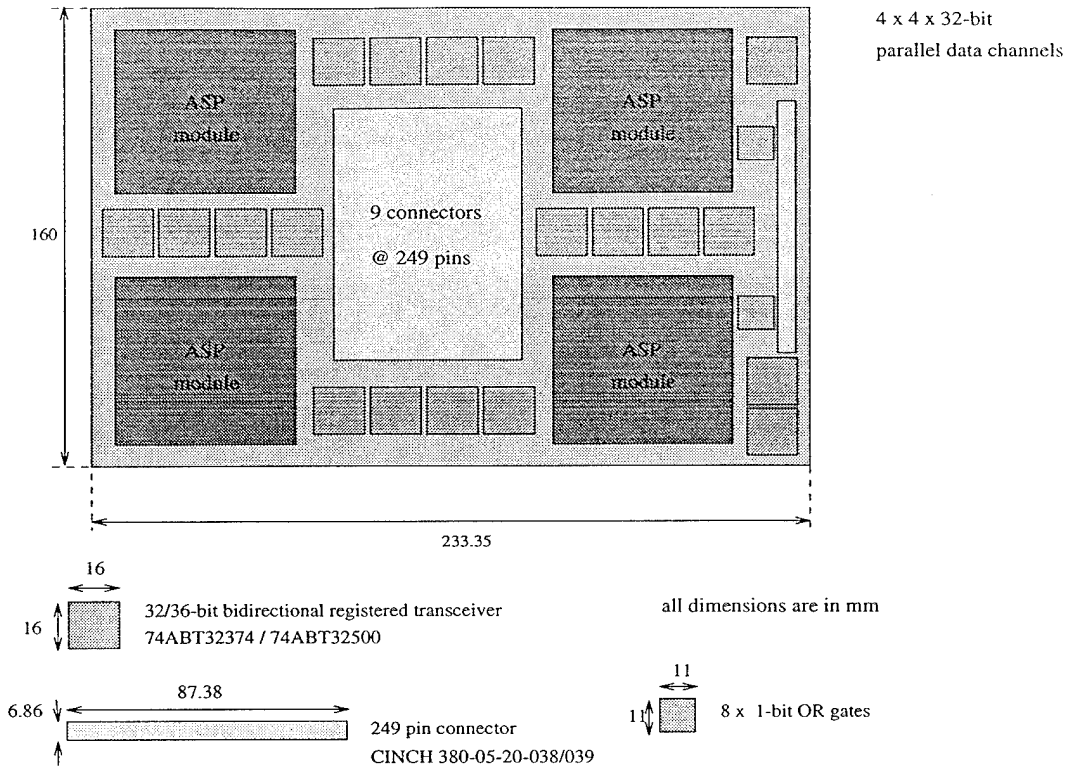

Figure 12: Modular-MPC MPP daughter-board

The resulting configuration options in terms of # PEs per I/O a channel are shown in Figure 14.

Two technological upgrades for the ASP modules are currently investigated by Aspex Microsystems Ltd.:

- 3-D chip stacks

  The first technology upgrade investigated explores the possibilities of stacking ASP chips. By mounting stacks of 4 ASP chips each on the HDI (see Figure 13) the number of APEs in an MCM can be increased by a factor of 4. Thus, the APEs on the four MPP daughter-boards of the presented design could be implemented on a single MPP daughter-board.

- 3-D ASP modules

  A further technology upgrade which is being investigated involves 3D ASP modules. These provide the possibility to integrate 64k APEs in a single package (i.e. a factor 16 increase compared with the current MCMs)

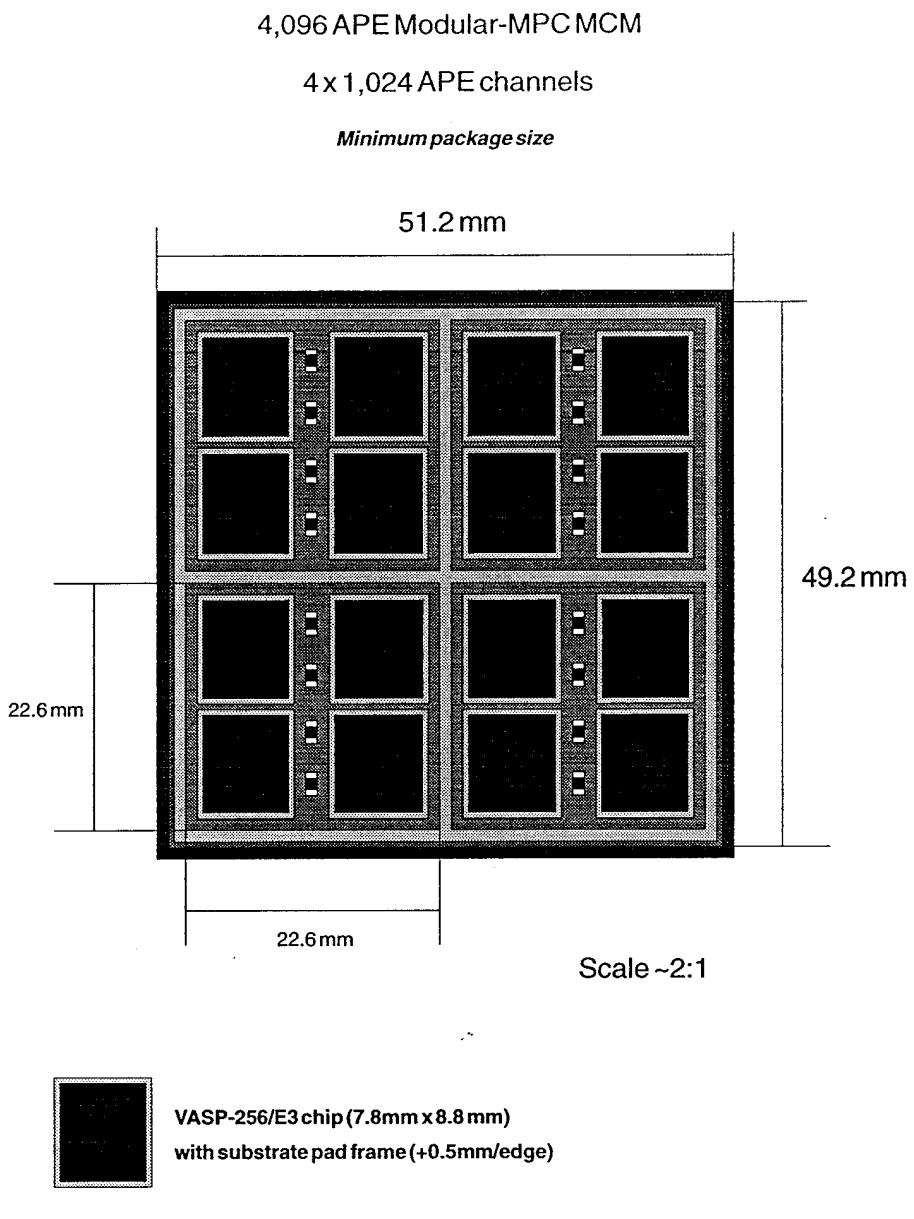

Figure 13: Modular-MPC MCM

4.2 DSM-SDS daughter-board

Figure 15 shows the layout of the DSM-SDS daughter-board. This board implements the SDS memory introduced in Appendix A.2.3.1. Thus only those configuration with the DSM configured as a 2-stage pipeline with more than 4 parallel data I/O channels (see section 4.4) or a 3-stage parallel data I/O pipeline (see Appendix A.2.3) include a DSM-SDS boards.

Each DSM-SDS includes up to 16 SIMMs. Each SIMM provides 32k, 64k, 128k or 256k 32-bit words storage. The total storage capacity of one DSM-SDS board thus is in the range of 32k to 4M 32-bit words, a system with 4 DSM-SDS boards could therefore accommodate 16M 32-bit words storage.

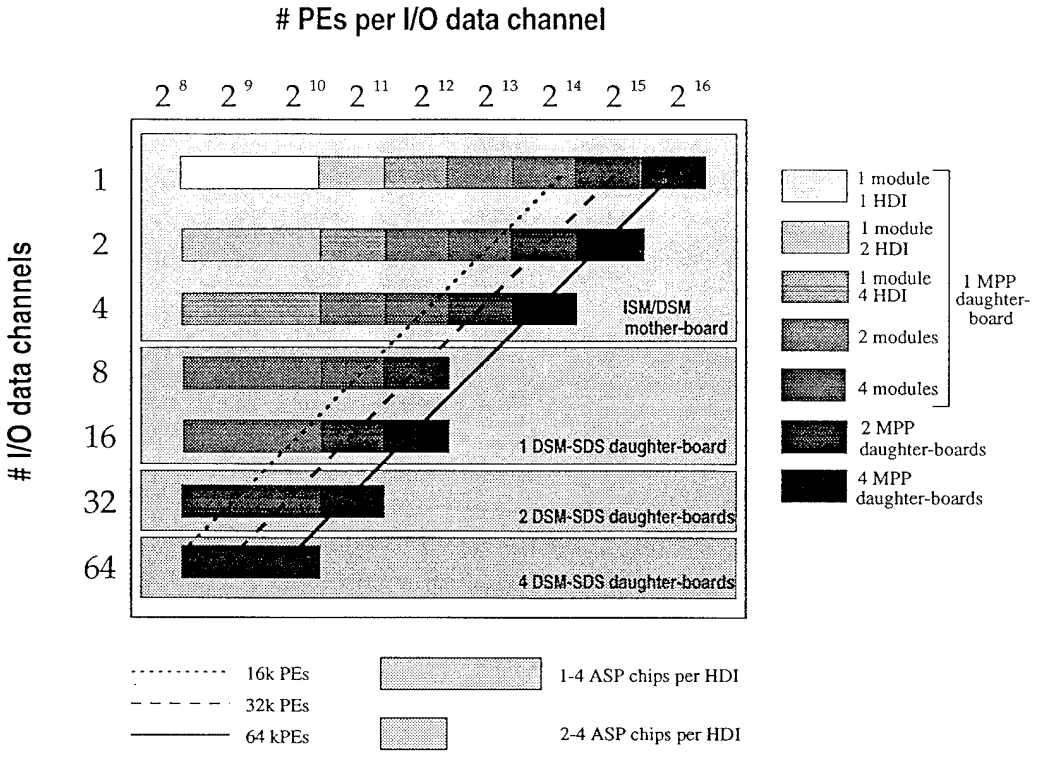

Figure 14: Modular-MPC scalability: # PEs per I/O data channel

The SDS configuration options in terms of memory size per I/O data channel are summarised in Figure 16.

The AND gates on the DSM-SDS daughter-board implement the memory module inhibitor introduced in Appendix A.2.3.1. As described in that section, the memory module inhibitor can operate in a data dependent way, such that only data with a specific signature is written to the memory modules. Due to the space restriction for this implementation of the Modular-MPC, this functionality could only be implemented in a minimal form, i.e. the Most Significant Bit (MSB) of a data item determines whether it is written to the memory module.

4.3 DSM-PDR daughter-board

The Parallel Data Router introduced in Appendix A.2.3.1.2 is implemented with up to 2 DSM-PDR daughter-boards. Thus only configurations with shared SDS (see Appendix A.2.3.1) will include DSM-PDR boards.

Each board can route 2-32 32-bit wide parallel data channels. The PDR is implemented as an n:n crossbar switch consisting of standard 64:64 bits routing chips (see [LSI94]).

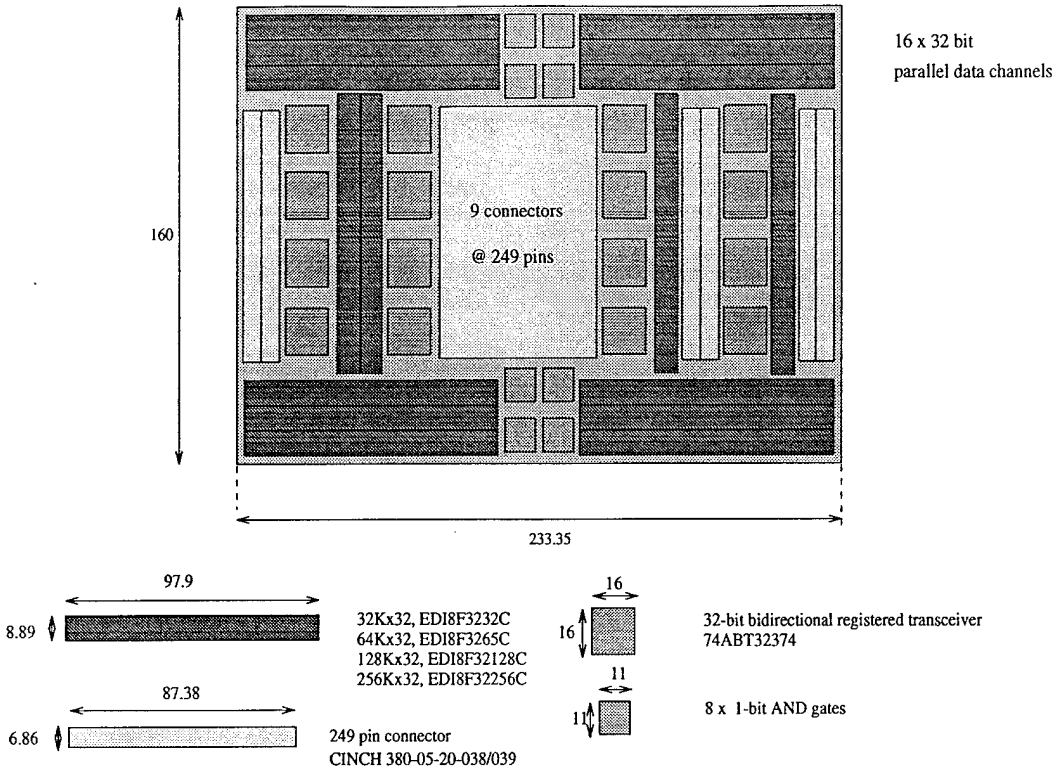

Figure 15: Modular-MPC DSM-SDS daughter-board

Figure 17 shows the router configuration for the routing of 32 parallel data channels. By assigning the same bit of each data channel to the same router chip, the number of required router chips grows linearly, rather than quadratically as might have been expected, with the number of routed data channels.

The configuration of the switches can be overlapped with data transfer, thus minimising potential inefficiencies caused by configuration times. Figures 18 and 19 show top and bottom of the DSM-PDR daughter-board. The drivers on the bottom of the DSM-PDR daughter-board are required to achieve the required data transfer rate for parallel data.

The PDR can be operated in 2 modes

- Unidirectional routing

  In this mode routing between different channels is only possible in one direction (either from the memory modules to the ASP modules *or* vice versa). Data transfers in the opposite direction can only be executed in a straight through manner, between associated ASP modules and memory modules. Data transfers in both directions are executed at 40 Mitems/sec (32-bit items).

  One DSM-PDR daughter-board implements the PDR for up to 32 parallel data I/O channels.

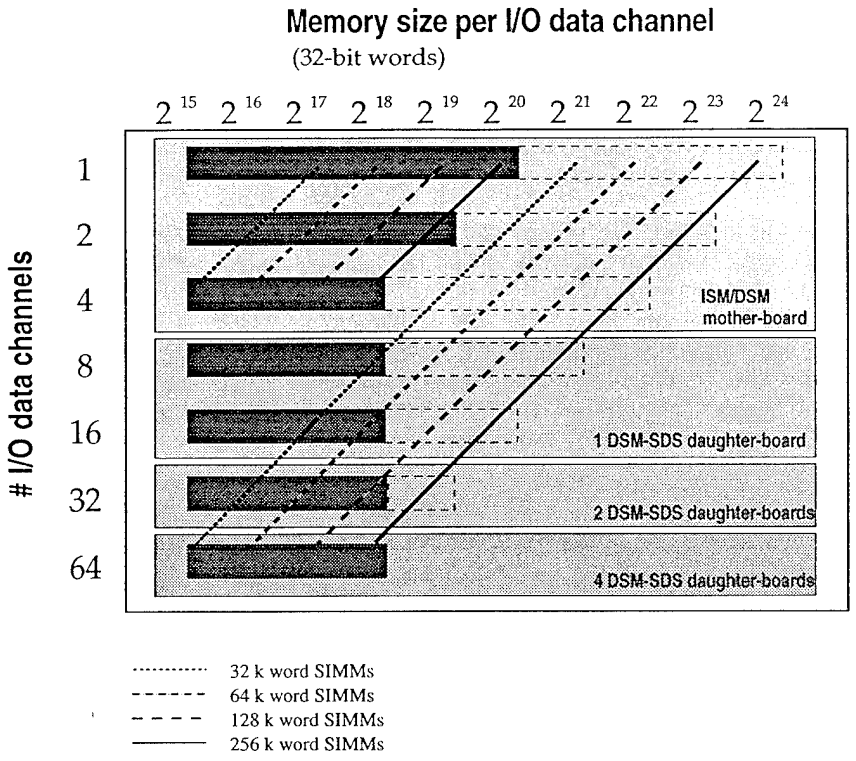

Figure 16: Modular-MPC scalability: Memory size per I/O data channel

- Bidirectional routing

  In this mode routing is possible in both directions (from the memory modules to the ASP modules *and* vice versa).

  Two speed options are available for this mode

  - 40 Mitems/sec (32-bit items) in one direction, 20 Mitems/sec (32-bit items) in the other direction

    One DSM-PDR daughter-board implements the PDR for up to 32 parallel data I/O channels.

  - 40 Mitems/sec (32-bit items) in both directions

    One DSM-PDR daughter-board implements the PDR for up to 16 parallel data I/O channels (i.e. 4 DSM-PDR daughter-boards are required to implement a PDR for 64 data channels).

4.4 ISM/DSM mother-board

The layout of the ISM/DSM mother-board is shown in Figure 20. It implements all functional units in the ISM (see Figure 32), the TDQ (see Appendix A.2.3.2), as well as the interfaces in

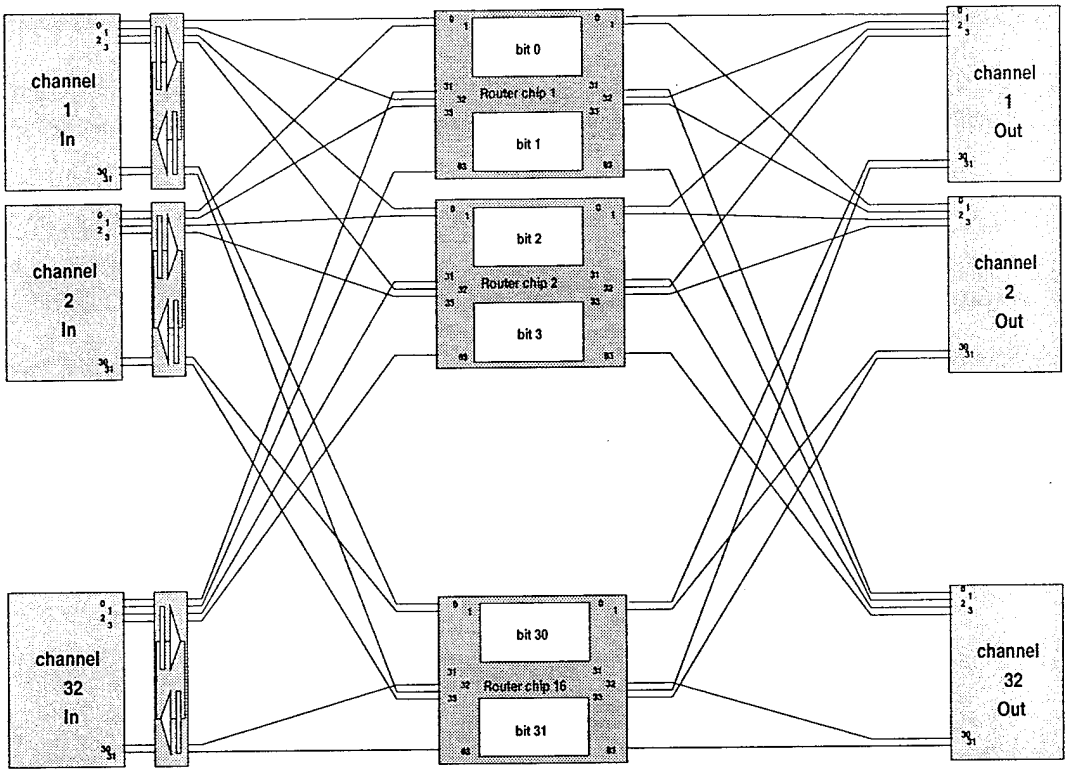

Figure 17: DSM-PDR configuration for 32 parallel I/O data channels

the DSM (see Appendix A.2.3). An external I/O interface (selected from a range of interfaces implementing different data transfer protocols) and a monitor interface can be mounted as mezzanine boards.

In the case where DSM-SDS boards (see section 4.2) are present, the ISM/DSM motherboard can implement the Tertiary Data Queue (TDQ, see Appendix A.2.3.2) of a 3-stage parallel data I/O pipeline in the DSM. The size of the TDQ depends on the chosen SIMM for the tertiary data buffer.

Since, for cost-effectiveness reasons single-port memory had to be used for the SIMMs in the parallel data path, SDTs and TDTs have to time share the access to the Secondary Data Store, i.e. all Secondary Transfer (SDT) has to be stopped while Tertiary Data Transfer (TDT) takes place. Therefore, in order to reduce the time for TDT to a minimum, the TDQ on the ISM/DSM mother-board is connected with 4 parallel data I/O channels to each DSM-SDS board.

If no DSM-SDS boards (see section 4.2) are present, the SIMMs which normally implement the TDQ can now be used as Secondary Data Store (SDS, see Appendix A.2.3.1). Thus, no DSM-SDS daughter-boards are required for all Modular-MPC configurations with the DSM being configured as a 1-stage parallel data I/O pipeline or a 2-stage pipeline with not more than 4 parallel I/O data channels for SDT (see Appendix A.2.3).

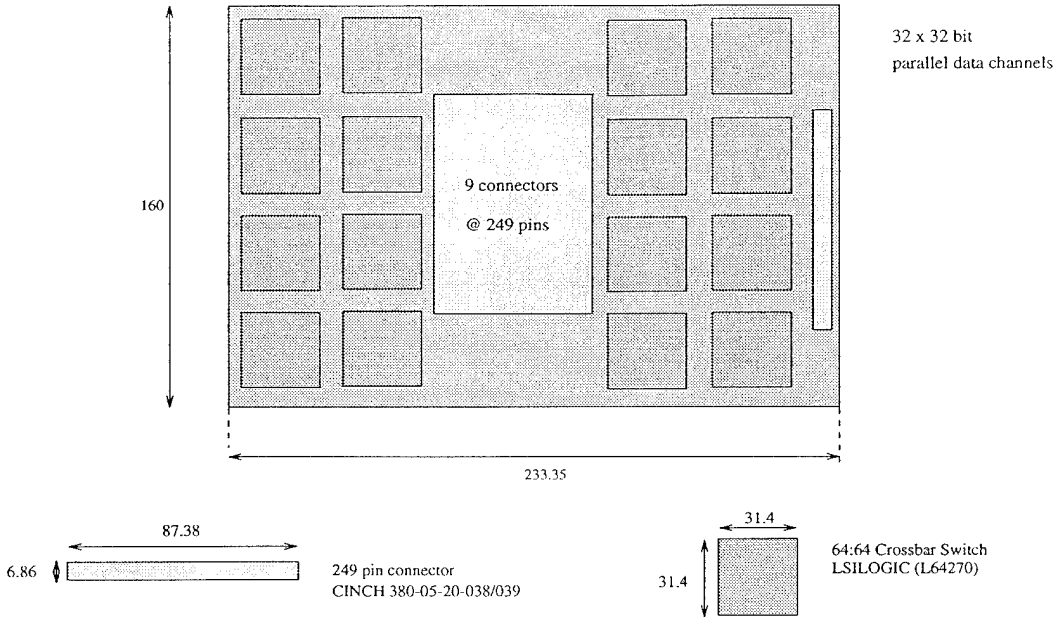

Figure 18: Modular-MPC DSM-PDR daughter-board (top)

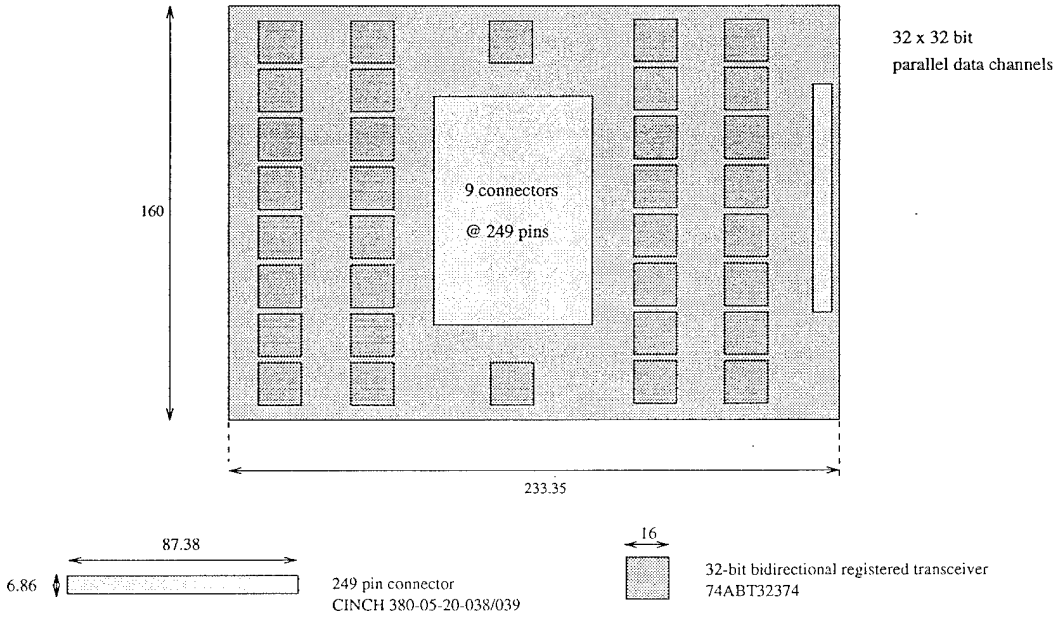

Figure 19: Modular-MPC DSM-PDR daughter-board (bottom)

As already mentioned above, the ISM/DSM mother-board provides, via the VME host-interface, the only connection of the Modular-MPC SIMD assembly to the host workstation. Note that the process scheduler (see Appendix A.2.4.1) and the micro controller (see Ap-

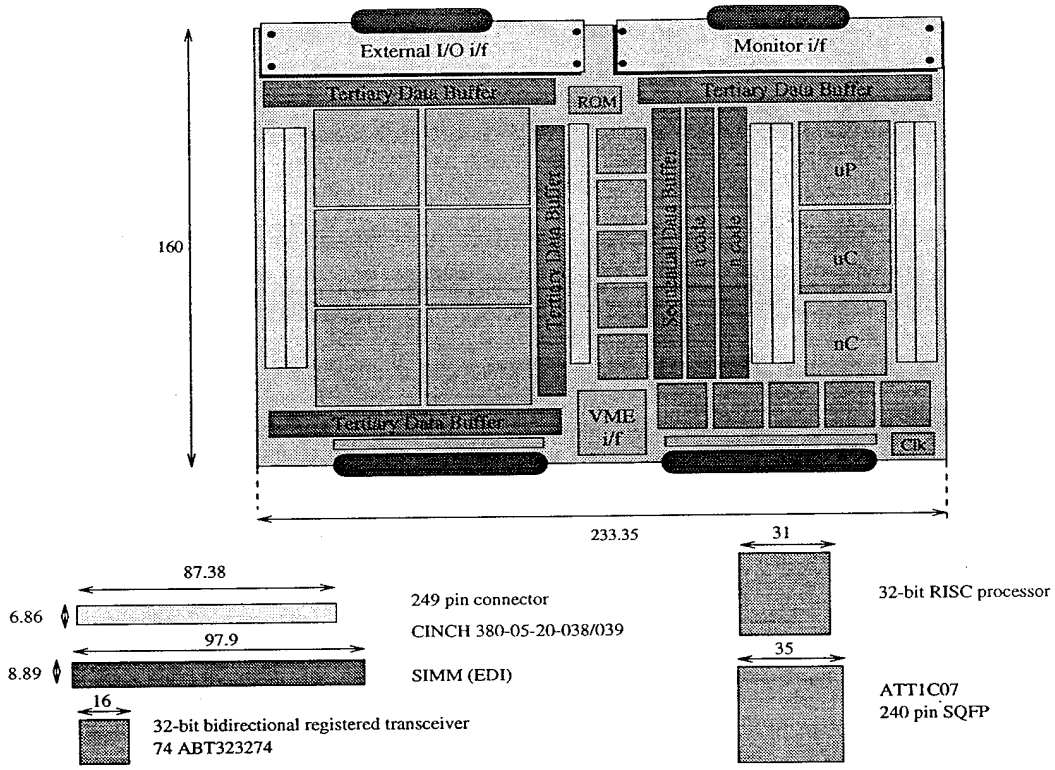

Figure 20: Modular-MPC ISM/DSM mother-board (fully populated)

pendix A.2.4.2) shown in Figure 32 can be integrated in a single 32-bit RISC processor. The nano controller (see Appendix A.2.4.2) is implemented with a Field Programmable Gate Array (FPGA).

The Sequential Process Execution Unit (SPEU, see Appendix A.2.4.3) is implemented with a 32-bit RISC micro processor with floating point unit and a SIMM which implements the Sequential Data Buffer (SDB).

Similarly, the parallel data I/O control is implemented with FPGAs (Field Programmable Gate Arrays). The considerable versatility of the I/O control demonstrated in Figure 33 can thus be implemented by downloading the required firmware modules into the FPGAs. The number of FPGAs can be minimised to the number required to implement particular modules of the I/O control.

Figure 21 shows a configuration of the ISM/DSM board which implements only a basic version of the SDT control, excluding the address generation. The FPGAs which implement the address generation are replaced by transceivers.

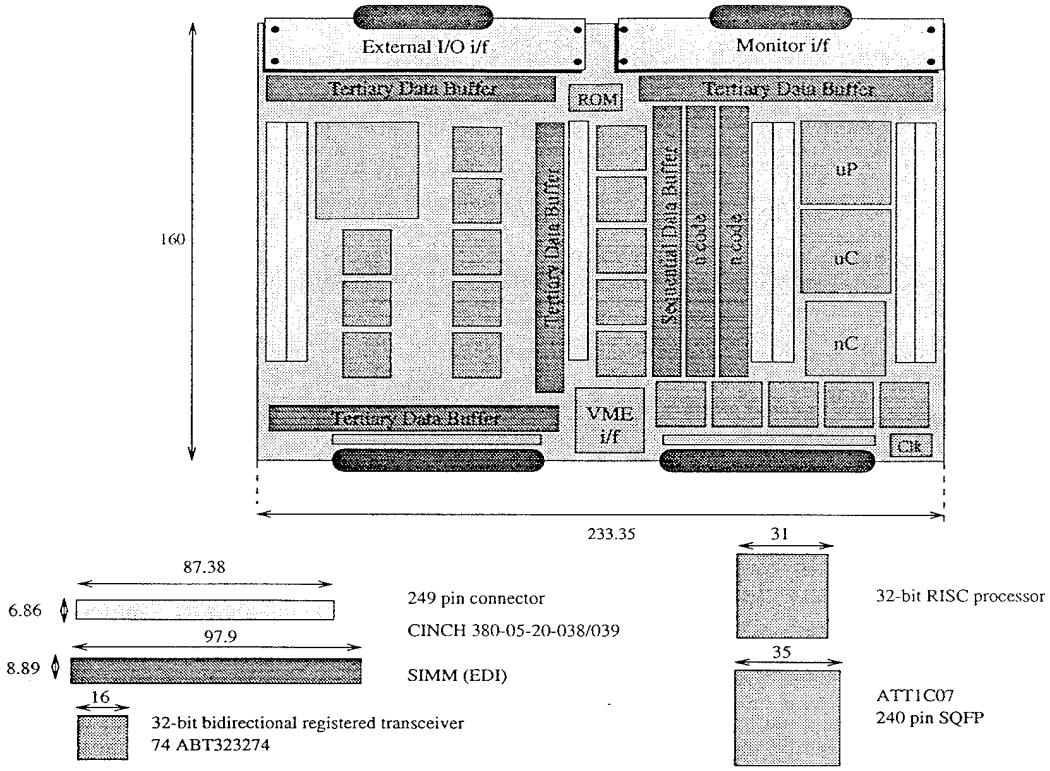

Figure 21: Modular-MPC ISM/DSM mother-board (partially populated)

5 Case study phase 2: Modular-MPC evaluation

The aim of this phase of the case study was to evaluate the application flexibility and cost-effectiveness for each application exemplar selected in section 3.2.1.

In the first part of this section the evaluation method used is introduced. In particular it is demonstrated how the Modular-MPC methodology introduced in Appendix A.1.1 was used to configure Modular-MPCs which match the natural parallelism of each exemplar. The second part of this section gives a short overview over the application requirements of each exemplar and presents the results of the evaluation.

5.1 Evaluation method

The evaluation of the Modular-MPC can be broken down into four steps, which were executed for each application exemplar.

5.1.1 Requirement analysis

In a first step the functionality of the required algorithm was analysed. Then, through close interaction with end-users of the application, requirements were derived. These included general requirements (e.g.framesizes, size of input data) and performance requirements. Experience gained during the project showed that some end-users tend to quote, as application requirements, what are actually requirements for certain solution strategies to the application. It was, therefore, important to carefully distinguish between 'real' and 'pseudo' application requirements.

5.1.2 Analysis of natural parallelism

Control-level and data-level parallelism was analysed through

- Identification of *task packages*, *tasks* and *processes*

  Having identified these basic building blocks of the algorithm, opportunities for control level parallelism were exposed with a flow-graph.

- Identification of *sub-images* associated with *tasks* and *task packages*, and *data-structures* associated with *processes*

  The sizes of the identified *sub-images* and *data-structures* directly exposed the opportunities for data-level parallelism.

5.1.3 Synthesis of applied parallelism

The synthesis of applied parallelism which was used in the evaluation of the Modular-MPC follows the scheme introduced in Appendix A.1.1. However, for the particular example for an instantiation of the Modular-MPC architecture, the Modular-MPC methodology used for the case-study can be explained in more detail.

5.1.3.1 Algorithm validation on ASP simulator After creating an ASP-specific implementation of each *process*, which exploits data-level parallelism and derives data structures by associative selection the algorithm was built. This was followed by functional validation using the ASP simulator [Asp90].

5.1.3.2 Application demonstration on ASTRA An application demonstration using the ASP test-bed (ASTRA [Asp93], [Asp94a]) provided functional validation in a systems context. Furthermore the parameterised processing time for each process was measured.

5.1.3.3 Modular-MPC configuration In order to configure the Modular-MPC to meet the application requirements cost-effectively, each of its main parts, MPP, ISM, DSM (see Appendix A.2), had to be configured, a decision had to be made which of the optional modules had to be included and how each module was to be configured.

- *Task package*, *Task* and *process* allocation

  In a first step the basic algorithmic building blocks identified in the analysis of the natural parallelism (see section 5.1.2) were assigned to execution units. Task packages were assigned to Task Execution Units (TEUs, see Appendix A.2.1). Depending on their natural data-level parallelism, *processes* were assigned either to the MPP (see Appendix A.2.2) or the Sequential Process Execution Unit (SPEU, see Appendix A.2.4.3). This allocation was also governed by the interdependencies of *processes*. Sequences of *processes* which could be executed on the same patch without the need to process other patches were identified.

- DSM configuration

  The Data Stream Manager (DSM, see Appendix A.2.3) was configured beginning with the Secondary Data Transfer (SDT, see Appendix A.2.3). For each stage of the parallel data I/O pipeline (see Appendix A.2.3 and Figure 28) a decision had to be made whether the application I/O requirements implicate the need for that stage.

  First the number of required parallel data I/O channels for SDT (see Appendix A.2.3) was determined. The 'balance point' (see section 2.4.3) between processing and I/O was calculated for each *process*, using the processing times calculated in section 5.1.3.2. Then a decision about the required number of data channels for SDT was made, taking cost-effectiveness issues into account.

  The need for and the size of the Secondary Data Store (see Appendix A.2.4.3) was then determined. A SDS is required if more than one parallel data I/O channel is used for the Secondary Data Transfer (SDT). The size of the SDS was based on the application requirements determined in section 5.1.1.

  The need for the Parallel Data Router (PDR, see Appendix A.2.3.1.2) evolved from an analysis of the data access patterns to the Secondary Data Store (SDS). Obviously, the size of PDR had to be adjusted to the number of parallel data I/O channels for SDT.

  Next, the need for and the size of the Tertiary Data Queue (TDQ, see Appendix A.2.3.2) was determined. For the Modular-MPC presented in the case-study, a TDQ is required to support parallel data I/O at a rate of more than 20 MBytes/sec.

  Finally, an interface for parallel data had to be selected. All parallel data I/O, at a rate of less then 4 MBytes/sec, can be handled by the VME interface (i.e. host interface, see section 4) of the HP747i. High-speed I/O or the use of specific data transfer protocols requires the use of an appropriate external I/O interface (mounted as mezzanine board on the ISM/DSM mother-board, see section 4.4). Applications which require a high-quality, high-speed graphics display make use of a monitor-interface (see section 4.4) necessary.

- MPP configuration

  The Massively Parallel Processor (MPP, see Appendix A.2.2) was configured determining the number of required PEs and the number of ASP modules over which they were distributed.

  The number of required PEs had to be based on the required time for each *process* of the application calculated as the maximum of I/O time and processing time. As already pointed out, processing time was based on the measurements of the ASTRA demonstration and projected for the Modular-MPC. The I/O time was based on the performance values of the standard components (e.g. SIMMs, drivers, router chips) involved in the data transfer and an electrical simulation of the board stack carried out as part of the design study (see section 4). Based on the performance requirements of the application, the maximal number of patches was calculated. Thus, the overall number of PEs was determined by the number of PEs required to achieve the calculated number of patches.

  Obviously, the number of ASP modules had to be chosen to match the number of parallel data I/O channels for SDT.

- ISM configuration

  The configuration of the Instruction Stream Manager (ISM, see Appendix A.2.4) could be broken down in three steps. First the need for the SPEU (see Appendix A.2.4.3) was determined, simply by checking whether any *processes* had been assigned in the first step of the Modular-MPC configuration. Then the parallel data I/O control was configured to match the configuration of the DSM. This process is described in some detail in Appendix A.2.4.4. Finally, based on the selected software modules and the application requirements, store sizes for the code stores and SDB (see section A.2.4.3) were selected.

5.1.3.4 Configuration optimisation Having derived a Modular-MPC configuration, it was, in an iterative manner, subjected to configuration optimisation until a satisfying result was achieved. In order to do that, a timing diagram based on the Modular-MPC configuration, which covered the interaction of all functional units in the Modular-MPC, was created for the application.

Based on this diagram three analysis steps were carried out

- high-level analysis

  The parallel computing inefficiencies (represented by $P_{oh}(0)$ and $P_{oh}(i)$ for small i, see section 2.4.1) apparent in the application were analysed. Bottlenecks were identified and analysed.

- detailed analysis

  A detailed performance analysis based on the timing diagram yielded the overall processing time for the application on the derived Modular-MPC configuration. The degree to which these parameters met the application requirements was analysed.

- cost analysis

  An overall overall system cost, based on costs quoted by suppliers for 100+ components, was determined.

Subsequently an optimisation governed by cost-effectiveness issues was carried out.

5.1.4 Evaluation

The evaluation was carried out for each application exemplar in two parts.

- comparison of the achieved applied parallelism with the natural parallelism of the application

  The main aim of this part of the evaluation was to determine how well the derived Modular-MPC configuration matched the natural parallelism. Points of efficiency loss were identified. In fact, the results of this evaluation phase were fed back into the design study (see section 4). Thus, in an iterative loop, problems emerging from the evaluation of the application exemplars were directly used for optimisation purposes.

- comparison of cost-performance values with other architectures

  Having compared the achieved applied parallelism to the absolute measure of natural parallelism in the first part of the evaluation, a 'relative' comparison, in respect to the cost-performance of other architectures, was attempted in the second part. Obviously, such comparisons can only give a rough guidance as to how well the Modular-MPC compares to other architectures, since the used assumptions will normally differ and are not always published.

5.2 Results

This section presents the results emerging from the evaluation of the Modular-MPC for each end-to-end application exemplar selected in section 3.2.1. (Note that the performance results for the selected task library exemplars (see section 3.2.1) will be presented in section 6). All steps discussed in section 5.1 were carried out for each exemplar. This section presents, for each end-to-end exemplar, the results of the requirement analysis (see section 5.1.1), the Modular-MPC configuration derived for the exemplar in section 5.1.3 and the achieved performance. For each exemplar, further discussion of the application requirements and the configuration of the Modular-MPC can be found in the cited references.

Note that the cost-performance comparison with other architectures is presented in the next section (section 6).

5.2.1 Near Earth Object Detection (NEO)

With mounting evidence of the cataclysmic consequences of an asteroid collision, there has been a growing concern for the vulnerability of the Earth to these interplanetary bodies or near-Earth objects (NEOs). In 1992, through the Spaceguard Survey report, recommendations were made on suitable scientific research and possible strategic defense measures to avoid such an occurrence. Using an integrated network of existing observatories fitted with large CCD-based sensor arrays, wide-field Schmidt telescopes would attempt to scan the near-Earth environment to discover Earth-approaching or Earth-threatening objects in real-time. Given a sequence of three input-images, the algorithm detects astronomical objects, extracts object characteristics, before object classification into static and non-static objects. Classification is achieved by the comparison of detected objects across the 3 images. Further information on the NEO application can be found in [Asp94b] and [Asp95b].

The following application requirements were determined:

input image:

$2048 \times 2048 \times 16$ bits image per CCD

performance requirements:

12 sec to process the input of 16 CCDs

Although a 2-stage solution including 2 TEUs (see Appendix A.2.1) was considered initially, the most cost-effective solution turned out to be a 1-stage solution, including a single TEU with **20k APEs**. The achieved performance is matched to the application requirements with **12sec to process the input of 16 CCDs**. The detailed Modular-MPC configuration is listed in Table 6.

Further information on the derived Modular-MPC configuration and its performance can be found in [Asp95b].

5.2.2 DARPA Image Understanding Benchmark II

The DARPA Image Understanding Benchmark II has been designed as a vision benchmark to evaluate parallel architectures. One of the major objectives of this benchmark is to evaluate the performance of different architectures for an integrated image interpretation task rather than compare performances for specific procedures. The benchmark consists of a model-based object recognition problem, given two sources of sensory input, intensity and

number of MPP daughter-boards number of APEs number of ASP modules	2 20 k 5
number of DSM-PDR daughter-boards number of routing chips	0 -
number of DSM-SDS daughter-boards number of SIMMs SIMM size	1 8 256k 32-bit words
external I/O interface monitor interface microprocessor in SPEU size of TDQ I/O controller	no no yes 0 32-bit words 2-stage configuration, distributed SDS

Table 6: Modular-MPC configuration derived for Near Earth Object detection

(noisy) range data, and a database of candidate models, which consist of a configuration of rectangular surfaces. As a solution to the recognition problem the model with the best match with the input data has to be identified. A detailed discussion of the DARPA Image Understanding Benchmark II and its results can be found in [WRHR91].

The application requirements of the DARPA Image Understanding Benchmark II are:

input images:

$512 \times 512 \times 8$ bits intensity image
$512 \times 512 \times 32$ bits depth image

performance requirements:

15-20 frames/sec

The derived Modular-MPC configuration for the DARPA Image Understanding Benchmark II comprises 1 TEU with **4k PEs**. This configuration has a performance of **18 frames per second**. The detailed Modular-MPC configuration is listed in Table 7.

number of MPP daughter-boards number of APEs number of ASP modules	1 4k 1
number of DSM-PDR daughter-boards number of routing chips	0 -
number of DSM-SDS daughter-boards number of SIMMs SIMM size	1 4 256k 32-bit words
external I/O interface monitor interface microprocessor in SPEU size of TDQ I/O controller	yes no yes 32k 32-bit words 3-stage configuration, distributed SDS

Table 7: Modular-MPC configuration derived for the DARPA Image Understanding Benchmark II

Further information on the derived Modular-MPC configuration and its performance can be found in [Asp95a].

5.2.3 Video signal compression

Digital video compression is an enabling technology for the impending revolution in communications and entertainment. It solves problems of economical storage, low speed transmission and bandwidth multiplication for applications such as multimedia, video-on-demand over telephone networks, direct broadcast satellite, cable television services and optical disks. Given the number of compression algorithm schemes - both proprietary and industry standard - the only viable solution is a programmable one. Moreover, many applications now have diverse requirements and call for more than one type of compression. In addition, depending on the applied video format and source data reduction factor, overall computation rates from 500 to 7000 MOPS are required for video processing.

Each task of a video compressing system can be characterised either as low level task, medium level task or control operation.

Low level tasks are Motion Estimation, FIR-filter, two-dimensional Discrete Cosine Transform (2D-DCT), two-dimensional Inverse Discrete Cosine Transform (2D-IDCT), Prediction and Inverse Prediction. Low level tasks operate on video data or, in the case of 2D-IDCT, spectral coefficients. Within each low level task every pixel or spectral coefficient of the input data is processed in exactly the same manner. Therefore, the low-level operations of the compression computational pipeline (which account for 85% of the overall computational rate) are suitable for parallel processing.

Medium level tasks are Quantisation, Inverse Quantisation and Variable Length Coding. Medium level tasks require a wide range of operations like division, variable thresholding, counting of events, minima evaluation and table-based search. Although the nature of medium level tasks is data dependent they demonstrate a high degree of parallelism because they are applied to a considerable number of data items. For example, although thresholding operations are based on dynamic and data-dependent adaptation of the threshold the task is applied to all output data of the DCT operation. Medium level tasks require approximately 10% of a compression system. The rest of the computational rate is required for control operations. For further information see [Asp95d].

The application requirements for MPEG data compression (CCIR standard 601 @ 30 MHz) are:

input data:

720×576 pixels luminance
$2 \times 360 \times 228$ bits chrominance

performance requirements:

30 frames/sec

The derived Modular-MPC configuration for video processing an compression comprises **64k APEs**. Its performance is **6.21 Gpixels for DCT/InverseDCT** and **76.7 Mvectors/sec for motion estimation**

The detailed Modular-MPC configuration is listed in Table 8.

Further information on the derived Modular-MPC configuration and its performance can be found in [Asp95d].

5.2.4 Volume rendering

Volume visualisation is a method for extracting meaningful information from volumetric data sets through interactive graphics and image processing techniques. It is a method for mapping volumetric scalar data field into an intensity map (i.e. image). An important feature of visualisation is to provide an understanding of object structures.

number of MPP daughter-boards number of APEs number of ASP modules	4 64 k 16
number of DSM-PDR daughter-boards number of routing chips	0 -
number of DSM-SDS daughter-boards number of SIMMs SIMM size	4 64 64k 32-bit words
external I/O interface monitor interface microprocessor in SPEU size of TDQ I/O controller	yes no yes 256k bytes 3-stage configuration, distributed SDS

Table 8: Modular-MPC configuration derived for MPEG data compression

On approach to volume visualisation is volume rendering. In this approach the volume is directly rendered. The first task is to rotate, scale and translate the object lattice until it fits the image lattice. Once the mapping has been carried out, the next step is to generate the shading. Finally the colour along the z-axis of the image lattice is composed to generate the image. Further information can be found in [Asp95e].

Application requirements which would satisfy users are:

input data:

$256 \times 256 \times 256 \times$ 8-bit model data

performance requirements:

25 frames/sec (real-time)

In order to achieve the performance requirements for volume rendering the maximum configuration of the Modular-MPC implementation introduced in section 4, comprising **64k APEs** is required. Depending on the rotation angle this configuration has a performance of 19 -

number of MPP daughter-boards number of APEs number of ASP modules	4 64k 16
number of DSM-PDR daughter-boards number of routing chips	1 8
number of DSM-SDS daughter-boards number of SIMMs SIMM size	4 64 256k 32-bit words
external I/O interface monitor interface microprocessor in SPEU size of TDQ I/O controller	no yes yes 256k 32-bit words 3-stage configuration, shared SDS

Table 9: Modular-MPC configuration derived for volume rendering

38.8 frames per second, its average performance is **25 frames per second.** The detailed Modular-MPC configuration is listed in Table 9.

Further information on the derived Modular-MPC configuration and its performance can be found in [Asp95e].

5.2.5 Surface rendering

In a second approach to volume visualisation, surface rendering, the model data is represented as a set of polygons which approximate the surface of the object. Thus, the transformation of the object into the image space is reduced to a transformation of polygon vertices. These are used in the subsequent rendering phase to compute colour and shading of each point in the final image. For further information see [Asp95c].

Application requirements which would satisfy users are:

input data:

1,000,000 polygons (120 bytes per polygon)

performance requirements:

1,000,000 polygons per second.

Initially a Modular-MPC configuration with separate stages for geometric transformation and rendering was considered. However, a configuration with a single TEU comprising **64k ASPEs** proved to be more cost-effective. This configuration can render **1.14 million triangles per second.**

The detailed Modular-MPC configuration is listed in Table 10.

number of MPP daughter-boards number of APEs number of ASP modules	4 64k 16
number of DSM-PDR daughter-boards number of routing chips	0 -
number of DSM-SDS daughter-boards number of SIMMs SIMM size	4 64 256k 32-bit words
external I/O interface monitor interface microprocessor in SPEU size of TDQ I/O controller	yes yes yes 256k 32-bit words 3-stage configuration, distributed SDS

Table 10: Modular-MPC configuration derived for surface rendering

Further information on the derived Modular-MPC configuration and its performance can be found in [Asp95c].

6 Case study phase 3: Comparison with other MPC architectures

This section presents the results of the final phase of the case-study. In this phase, the aim was to attempt a cost-performance comparison between the results gained for the Modular-

MPC (see section 5) and other architectures aiming at high-performance image processing.

6.1 End-to-end application exemplars

The task of finding published performance quotes for end-to-end application exemplars proved to be rather difficult. In fact, enough information to attempt a comparisons could only be gained for volume rendering (see section 5.2.4) and surface rendering (see section 5.2.5). An initial performance comparisons for the DARPA Image Understanding Benchmark II can be found in [WRHR91].

Table 11 shows the performance comparison for volume rendering (source: [Asp95e]).

	# PEs	framerate (frames/sec)
TMI CM-5	512	0.22
MasPar MP-2	16k	0.40
PVM	32	0.07
Fujitsu AB-1000	128	0.018
IBM PVS	8	0.66
Princeton Engine	1024	2.6
SGI Challenge	16	4.0
Modular-MPC	64	25

Table 11: Performance comparison volume rendering

As described in section 5.2.4 the Modular-MPC can be configured to execute volume rendering at a framerate of 25 frames/sec. In contrast, most of the other architectures have a performance of well below 1 frame/sec. In fact, the Modular-MPC is more than 6 times faster than the fastest solution available to date, the SGI Challenge, which includes special purpose hardware dedicated to volume rendering.

Table 11 shows the performance comparison for surface rendering (source: [Asp95c]).

machine	performance in (triangles per second)
SGI Reality Engine 2	930 k
Division Pixel Plane 6	324k
E&S Freedom 6000	800 k
Sun Microsystem Leo	210 k
Modular-MPC (64k PEs)	1.14 M

Table 12: Performance comparison surface rendering

In the case of surface rendering, performance data was only available for special purpose hardware dedicated to surface rendering. However, even in comparison with these machines, the performance of the Modular-MPC is outstanding.

6.2 Image processing tasks

Although, no performance data for end-to-end image processing exemplars could be found for other MPCs aimed at image processing, more information was available regarding their performance for specific image processing tasks. This section lists the results for the comparisons of a Modular-MPC with the following architectures (each of which is assumed to be configured in its maximum configuration):

- CNAPSE Server II [Ada94]

  The CNAPSE Sever II is developed by Adaptive Solutions Inc. Performance is quoted from [Bak91] for a CNAPSE Server II in its full configuration with 512 PNs.

- DAP Gamma [Cam94]

  The DAP Gamma is developed by Cambridge Parallel Processing. Performance figures are quoted for a DAP-4000 with 4096 processors.

- MaxVideo 200 [Dat94]

  The MaxVideo 200 is developed by Datacube. Performance values are quoted for a fully configured system.

Two configurations of the Modular-MPC were considered:

- 16k APEs

  configured with 1 ISM/DSM mother-board (see section 4.4) and 1 MPP daughter-board (see section 4.1)

- 64k APEs

  configured with 1 ISM/DSM mother-board (see section 4.4) and 4 MPP daughter-boards (see section 4.1)

Note that the performance results quoted in this section do not include any time required for I/O, although this time might be very significant. In fact, for many architectures it is the main source for parallel processing overheads represented by $P_{oh}(0)$ (see section 2.4.1).

Table 13 shows a comparison for a convolutions with different mask sizes on a 512×512 image.

mask size	Modular-MPC 16k APEs	Modular-MPC 64k APEs	CNAPSE	DAP Gamma	MV 200
3×3	0.392	0.098	3.61		15
5×5	1.04	0.26			
7×7	1.96	0.49	9.55		
9×9	3.33	0.833			
15×15	15.48	3.87	31.2		

Table 13: Performance comparison convolution on 512×512 image (processing time in ms)

Table 13 demonstrates that for the listed convolutions the Modular-MPC with 64k APEs is more than an order of magnitude faster than any of the architectures it was compared with, although all of these architectures have PEs with higher processing power than an APE in the Modular-MPC. This also applies for the Modular-MPC with 16k APEs which 2-10 times faster than all other architectures.

Similar results can be gained for a comparison of other image processing tasks, listed in Table 14.

	Modular-MPC 16k APEs	Modular-MPC 64k APEs	CNAPSE	DAP Gamma	MV 200
sobel filter (3×3 mask)	0.744	0.186		13.6	2.4
median filter (3×3 mask)	0.632	0.158		4	13.6
thresholding	0.14	0.035	0.308	1.6	
histogramming	88.2	22.05		65	19.8
normalise	0.1024	0.0256		1.6	
2D FFT	514.8	128.7	239.1		
jpeg compression	107.2	26.8	33.0		

Table 14: Performance comparison image-processing tasks on 512×512 image (processing times in ms)

Table 14 demonstrates, that the Modular-MPC with 64k APEs is always faster than all other architectures used in the comparison. However, this does not apply Modular-MPC with 16k APEs, which is slower than other architectures for some image-processing tasks (i.e. histogramming, 2D FFT and jpeg compression).

7 Conclusions

This document presented the results of an investigation into cost-effective accelerators for image processing workstations, carried out as part of phase 1 of the ARPA AVIS project.

The application requirement analysis (section 2) in the first part of the document exposed a very diverse range of image processing applications (see Table 1). In fact, it became clear, that in order to provide enough *application-flexibility* to cover the whole range of image processing applications *cost-effectively*, accelerators for image processing workstations have to meet a number of criteria (see Table 3):

- application flexibility
  - machine versatility
  - user acceptability
  - performance scalability
- cost-effectiveness
  - size, weight, power constraints
  - cost limitations
  - operational efficiency
  - future proofing

Image processing workstations cannot meet these requirements, especially in terms of performance scalability. The only apparent way towards meeting performance requirements of image processing applications is to make use of parallelism. Fortunately, image processing applications often show massive data-level parallelism together with modest control-level parallelism. However, looking at current MPCs, which attempt to make use of this performance potential, it became clear that these failed to meet requirements outlined in Table 3.

The Modular-MPC architecture presented in Appendix A is designed to overcome the difficulties besetting current MPCs. Therefore an investigation was carried out with the aim to test the (theoretical) Modular-MPC concept under real constraints in terms of *application flexibility* and *cost-effectiveness* (see Table 3). In order to design this investigation as realistic as possible a case study was set up as a practical test of a particular implementation of the Modular-MPC architecture. The HP 747i image processing workstation [Hew93] was selected as a vehicle for this study. Furthermore application exemplars covering the whole range of image processing applications were selected for an evaluation study. The case study was then carried out in three phases, a Modular-MPC design study within the restrictions of the HP747i workstation (see section 4), an evaluation of the *application-flexibility* and the

cost-effectiveness, based on the selected application exemplars, of the particular implementation defined in the case-study and, finally, an attempt of a cost-effective comparison with other architectures.

An implementation of the Modular-MPC architecture was found which meets the constraints in size, power and cooling imposed by the HP 747i workstation (see section 4). In fact, the résults of design study were presented to and endorsed by engineers at Hewlett Packard. Furthermore it could be shown that the Modular-MPC meets requirements outlined in Table 3.

- application flexibility
  - machine versatility

    The Modular-MPC was subjected to the whole range of image processing applications. It could be configured to match the natural parallelism of all application exemplars and met all application requirements (see section 5.2).
  - user acceptability

    The Modular-MPC can be implemented in a workstation environment, if necessary transparent to the user (see section 4). The software architecture of the Modular-MPC (see Appendix A.3) provides a number of different means to accesses the Modular-MPC, covering the whole range of user requirements - from total transparency of the Modular-MPC to full control over the Modular-MPC hardware.
  - performance scalability

    Performance scalability was demonstrated with the application exemplars (see section 5.2), where the Modular-MPC proved to be flexible enough to adapt to varying performance requirements.
- cost-effectiveness
  - size, weight, power constraints

    The design study showed that the Modular-MPC can meet the constraints imposed by the HP 747i workstation with currently available components (see section 4).
  - cost limitations

    The Modular-MPC appears to be well below the cost-limits of $20k for the entry level configuration and $100 k for the most sophisticated configurations. In fact, the calculated procurement cost were within a factor 2-3 below those limits. However, no attempt has been made to include overheads (marketing, support etc) in the cost calculation.
  - operational efficiency

    The operational efficiency was adequately demonstrated by the exemplars (see section 5.2).

- future proofing

  Apart from the future proofing inherited from the standard components a route to a factor four technology upgrade of the ASP modules can be seen (see section 4.1).

In view of the excellent cover of the application requirements shown above, it can be concluded that the Modular-MPC is feasible and meets all criteria outlined in Table 3.

The investigation also showed that the Modular-MPC compares very favourably with other MPCs (see section 6).

From the data which was available for other MPCs, it emerged that the Modular-MPC compared well in terms of the performance of *task library modules* (see section 6.2). In fact, for all exemplars of image processing tasks for which data was available, the Modular-MPC with 64k APEs performs, on average an order of magnitude, better than the architectures which were used as a comparison. However, three image-processing task exemplars could be found where the minimum configuration of the Modular-MPC with 16k APEs, is not as fast as some of the architectures (in their maximum configuration) it was compared to.

Even more impressive results were gained for the selected *end-to-end application exemplars* (see sections 5.2 and 6.1). As evident in Table 11, the performance results gained for volume rendering were exceptional. Not only does the proposed implementation meet the demanding performance requirements of real-time rendering of a $256 \times 256 \times 256$ voxel model, this performance is also outstanding compared with any other approaches, including special purpose hardware.

Although not as outstanding as the results for volume rendering, the performance projections for surface rendering are also very impressive. With the proposed configuration (Table 10), it was possible to surpass the 1,000,000 triangles/sec mark. Thus, the performance is comparable to the performance of the most powerful special purpose hardware (see Table 12).

Similarly encouraging are the results for the DARPA Image Understanding Benchmark II. In fact, it could be shown that the Modular-MPC delivers a much better performance than a previous ASP submission, which was the 'winner' of the benchmark (see [WRHR91]).

Unfortunately, comparison for end-to-end application exemplars was mainly restricted to special-purpose hardware. Data for end-to-end application performance of MPC architectures (which have been considered for the comparison of task library performance) was not available. In view of the importance of end-to-end application performance, this is particularly disappointing.

Thus, considering the results of all three phases of the case-study, it can be concluded that the Modular-MPC is a suitable candidate for cost-effective image processing workstation accelerators.

The project demonstrated the significance of large number of cheap data channels. In fact, two of the investigated application exemplars (MPEG data compression (see section 5.2.3) and surface rendering (see section 5.2.5)) required 64 parallel I/O data channels. Thus, the flexible, highly scalable solution to the I/O bottleneck employed by the Modular-MPC, namely the policy of providing a large number of cheap channels which can be populated with PEs and memory when required, paid off.

However, providing a large number of data channels makes board-to-board interconnection a critical issue. Nevertheless, a solution was found in the design study (see section 4) with existing connectors (i.e. CIN::APSE [Chi91]). Improvements in connector technology could be incorporated directly in the design.

A further point, which was emphasised by the results of the study, is the significance of three levels of routing used in the Modular-MPC. Three different networks serve three different purposes. The IACN (see Appendix A.2.2) is provided for activity passing, the PDR (see Appendix A.2.3.1.2) handles data reorganisations on-the-fly and the ITPDR (see Appendix A.2.3.3) is dedicated to data movement between task execution units. Thus, network functionality is provided where it is required as opposed to other architectures which implement all functionality in one complex, expensive network. This fact is particularly exposed for volume rendering (see section 5.2.4). The performance results demonstrate that, due to the Parallel Data Router (see Appendix A.2.3.1.2), the Modular-MPC succeeds where other MPCs fail because of the severe overheads caused by data reorganisation.

The fact, that the rather complex ASP structure is delivered encapsulated in a single unit, the ASP module MCM (see Appendix 4.1), proved to be a major advantage for simplicity in configuration and upgradability. A large number of APEs could be integrated in the Modular-MPC, thus reducing the overheads caused by patching (see section 2.4.2) significantly. As discussed in section 4.1 a clear route for technology upgrades of the ASP modules has been identified. It is possible to increase the number of APEs on a MPP daughter-board by a factor of 4, by encapsulating 3D chip stacks in a MCM. In fact, the next generation of MCMs, and the one beyond, can already been envisioned (and are actively studied).

The application exemplars demonstrated, yet again, the remarkable flexibility of ASP. It met different application requirements very well and due to its infinite scalability it is on par with standard components.

The case study also demonstrated the significance of the evaluation for end-to-end application exemplars. In fact, during the evaluation phase described in section 5 it became clear that most problems which contribute to the parallel processing overheads $P_{oh}(0)$ (see section 2.4.1) do not show in a library module evaluation. Therefore, it is particularly disappointing that, apart from special purpose hardware, data for end-to-end application performance were hardly available.

Another result of the case study was that, although several application exemplars showed modest control-level parallelism (see section 5.2.1 and 5.2.5), it was more cost-effective to implement them on a single TEU (see Appendix A.2.1). This seems to indicate, that low number of TEUs, probably not more than 2, is sufficient for cost-effective implementations.

Finally, it could yet again be demonstrated that the approach to parallel programming via library access (see Appendix A.3) is viable and efficient way to implement parallel processing applications.

Overall, the Modular-MPC demonstrated unprecedented scalability. In fact, its scalability was very impressive even within the constraints of HP747i, up to 64k APEs, up to 64 parallel data I/O channels and up to 64 MBytes of data storage for patching (see section 4). Other implementations with relaxed size and power restriction can provide even bigger scalability range.

The project presented in this report was rather ambitious. Subjecting the Modular-MPC paradigm to real constrains required a major design study activity. A lot of effort went into detailed electrical, cooling and cost analyses. Somewhat aggressive technology was used, however, for specific configurations the technology can be relaxed. In fact, the decision to use the HP 747i workstation as a vehicle for the case-study paid off in retrospect, since it made the design study much more realistic. However, since the successor of the HP 747i, the HP 748 i [Hew94] is now available, this workstation would be, due to its relaxed space restriction, a better vehicle for the prototyping of the Modular-MPC.

The results of this project show a clear route to market for the Modular-MPC. In terms of the Modular-MPC hardware, it would be wise to work closely with a manufacturer (e.g. Hewlett Packard). A ladder of cooperation could span from technical support, endorsement of the product and finally to the inclusion of the Modular-MPC as a catalogue item.

In terms of user-acceptability, several groups of customers can be identified, as shown in Figure 22:

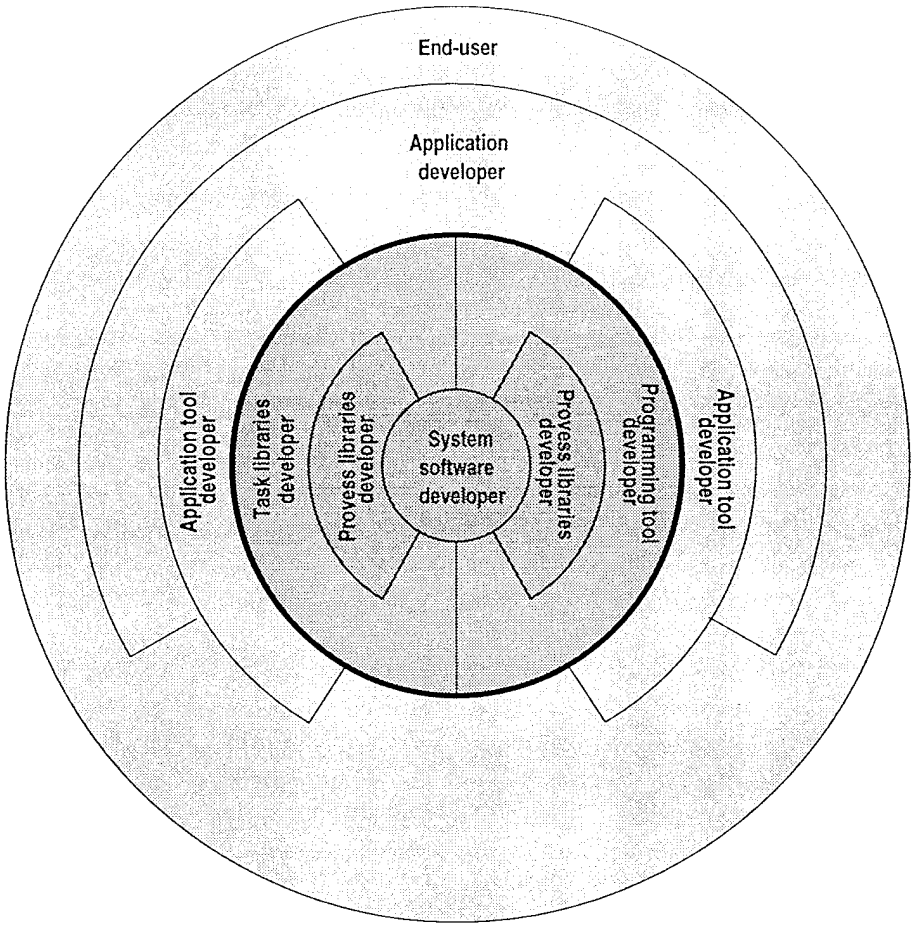

Figure 22: Modular-MPC users

- end-users
- application developers, who create applications for end-users and use application-tools
- application tool developers, who create application-tools for application developers with the help of programming tools and task libraries
- programming tool developers, who provide sequential programming environments, tailored to the needs of different groups of users
- task library developers, who build application-specific application libraries for application-tool and application developers
- process libraries and system software developers

In view of the encouraging results of this study, the next steps should be to put effort into turning the design study into an implementation. Application development, going hand in

hand with a formal specification of the Modular-MPC hardware and system software, should aim at the most promising application areas:

- volume rendering for medical and military applications
- image analysis for automatic target recognition
- application tool implementation (e.g. Analyze [RH90], Photoshop [GS95])

References

[Ada94] Adaptive Solutions Inc., 1400 N.W. Compton Drive, Suite 340, Beaverton, OR 97006, USA. *CNAPSE Server II Data Sheet*, 1994.

[Ame93] American National Standard for Information Systems. *High-Perfromance Parallel Interface - Framing Protocol (HIPPI-FP)*, X3T9..3 edition, March 1993.

[Ame94] American National Standard for Information Systems. *SCSI-3 FAST-20 (working draft)*, X3t10/1071D edition, 1994.

[Asp90] Aspex Microsystems Ltd., Brunel University, Uxbridge UB8 3PH, UK. *User's Manual for Release 2.0 & 3.0 of the VASP-SIM Simulator*, 1.2 edition, Nov 1990.

[Asp93] Aspex Microsystems Ltd., Brunel University, Uxbridge UB8 3PH, UK. *ASTRA Application Programmer's Reference Manual*, 1.0 edition, June 1993.

[Asp94a] Aspex Microsystems Ltd., Brunel University, Uxbridge UB8 3PH, UK. *ASTRA Application Programmer's course*, 1.0 edition, March 1994.

[Asp94b] Near-Earth Object detection using the Moduar-MPC: Requirements definition. Technical report, Aspex Microsystems Application Division Internal Report, Brunel University, Uxbridge, UB8 3PH, UK, June 1994.

[Asp95a] DARPA Image Understanding Benchmark II using the Modular-MPC. Technical report, Aspex Microsystems Ltd. Applications Division Internal Report, Brunel University, Uxbridge UB8 3PH, UK, Feb 1995.

[Asp95b] Near-Earth Object Detection using the Modular-MPC. Technical report, Aspex Microsystems Ltd. Applications Division Internal Report, Brunel University, Uxbridge UB8 3PH, UK, Feb 1995.

[Asp95c] Surface Rendering using the Modular-MPC. Technical report, Aspex Microsystems Ltd. Applications Division Internal Report, Brunel University, Uxbridge UB8 3PH, UK, Feb 1995.

[Asp95d] VADIS EU-601. Technical report, Aspex Microsytems Ltd., Brunel University, Uxbridge UB8 3PH, UK, Jan 1995.

[Asp95e] Volume Visualisation using the Modular-MPC. Technical report, Aspex Microsystems Ltd. Applications Division Internal Report, Brunel University, Uxbridge UB8 3PH, UK, Feb 1995.

[Bak91] T. Baker. Artificial Neural Netwrok and Image Processing using the Adaptive Solutions' Architecture. Technical report, Adaptive Solutions, Inc., 1400 N.W. Compton Drive, Suite 340, Beaverton, Oregon, 97006, USA, March 1991.

[Cam94] Cambridge Parallel Processing, 16755 Von Karman Av, Suite 120, Irvine, CA 93714, USA. *DAP GAMMA-1000 and DAP GAMMA-4000*, 1994.

[Chi91] Chinch Connectors Ltd., 1500 Morse Av., Elk Grove Village, Illinoise 60007. *CINCH Data CIN::APSE Stacking Connector*, 1991.

[Dat94] Datacube, 300 Rosewood Drive, Danvers, MA 01923, USA. *Max Video 200 Target/Development System*, 1994.

[GS95] A.D. Greenberg and Greenberg S. *Fundamental Photoshop: A complete Introduction.* Osborne McGrawHill, 2nd edition, 1995.

[HD92] P.B. Hefferhan and D. Dekel. Imaging applications platform: concept to implementation. In *Proc. SPIE*, pages 495–509, 1992.

[Hew93] Hewlett Packard. *HP 9000 Series 700i, Models 747i/50, 747i/100*, 1993.

[Hew94] Hewlett Packard. *HP 9000 Series 700i/rt Models 743i, 743rt, 748i*, 1994.

[Lea88] R.M. Lea. ASP: A cost-effective Parallel Microcomputer. *IEEE Micro*, (10):10–29, Oct 1988.

[LSI94] LSI Logic Databook. *L64270 64 to 64 Crossbar Switch*, 1994.

[Res94] Research Systems Inc., 777 29th Street, Suite 302, Boulder, Col. C080303, USA. *Interactive Data analyis Language (IDL) Reference Guide*, 1994.

[RH90] R.A. Robb and D.P. Hanson. ANALYZE: A software system for biomedical image analysis. *Proc. of the 1st Conference on Biomedical Image Analysis*, pages 507–518, 1990.

[WRHR91] C. Weems, E. Riseman, A. Hanson, and A. Rosenfeld. THE DARPA IMage Understanding Benchmark for Parallel Computers. *JPDC*, (11):1–24, 1991. description and results of the DARPA banchmark.

A Modular-MPC architecture

This section introduces, the basic Modular-MPC concept, its hardware architecture as well as its software architecture, in detail.

A.1 Modular-MPC concept

The Modular-MPC architecture is aimed at satisfying the application requirements spelled out in section 2.1. In particular it is designed to overcome the deficiencies which have been identified for other MPC approaches (see Table 4).

- application flexibility
  - *Machine versatility* as well as *performance scalability* is achieved through application-specific configurations of generic hardware modules and software modules (see Appendix A.1.1 and section 5.1) in order to match the natural parallelism (see section 2.3) of the application.
  - *User acceptability* can be gained by providing many different user-environments which adapt to the specific needs of different users. (see Appendix A.3)
- cost-effectiveness
  - *Size, weight, power* and *cost* are, compared to current MPCs, significantly reduced through the consequent use of microelectronics (see Appendix A.2 and section 4). Application specific configurations are configured from *generic* modules, which can be mass-produced and are based on mass-produced microelectronics components (i.e. RAMs, microprocessors, FPGAs and ASP modules)
  - *Operational* efficiency is secured through Modular-MPC functionality and the efficiency of ASP modules (see Appendix A.2).
  - *Future proofing* of the Modular-MPC is "inherited" from the technology roadmap for high-volume off-the-shelf components (e.g. memories, microprocessors, FPGAs) and ASP module technology upgrades (see section 4).

A.1.1 Modular-MPC methodology

The Modular-MPC methodology is based on the following steps, shown in Figure 23, carried out for the particular requirements (e.g. framesize and framerate) of each application.

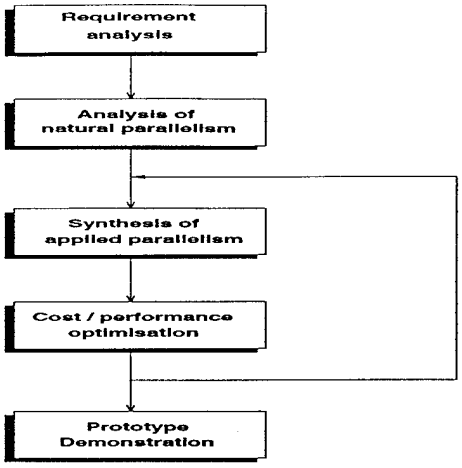

Figure 23: Modular-MPC methodology

- Analysis of the natural parallelism

  Task packages, *tasks* and *processes* (see section 2.3) are identified. A flow-graph exposes opportunities for control-level parallelism. Subsequently *sub-images* associated with *task packages* and *tasks* as well as *data-structures* associated with *processes* (see section 2.3) are identified. The size of the *sub-images* and *data-structures* exposes opportunities for data-level parallelism.

- Synthesis of the applied parallelism

  Modular-MPC software modules are installed in order to match the identified *tasks* and *processes*. The algorithm is then functionally verified on a general Modular-MPC. Subsequently a configuration of hardware modules for a specific Modular-MPC which matches the natural parallelism is derived and optimised.

A.2 Hardware architecture

Figure 24 shows the high-level architecture of the Modular-MPC.

It is partitioned into the three main functional blocks introduced in section 2.4:

- Massively Parallel Processor (MPP), where parallel processing takes place
- Data Stream Manager (DSM), which supports parallel data transfer
- Instruction Stream Manager (ISM), which supports sequential data transfer and control

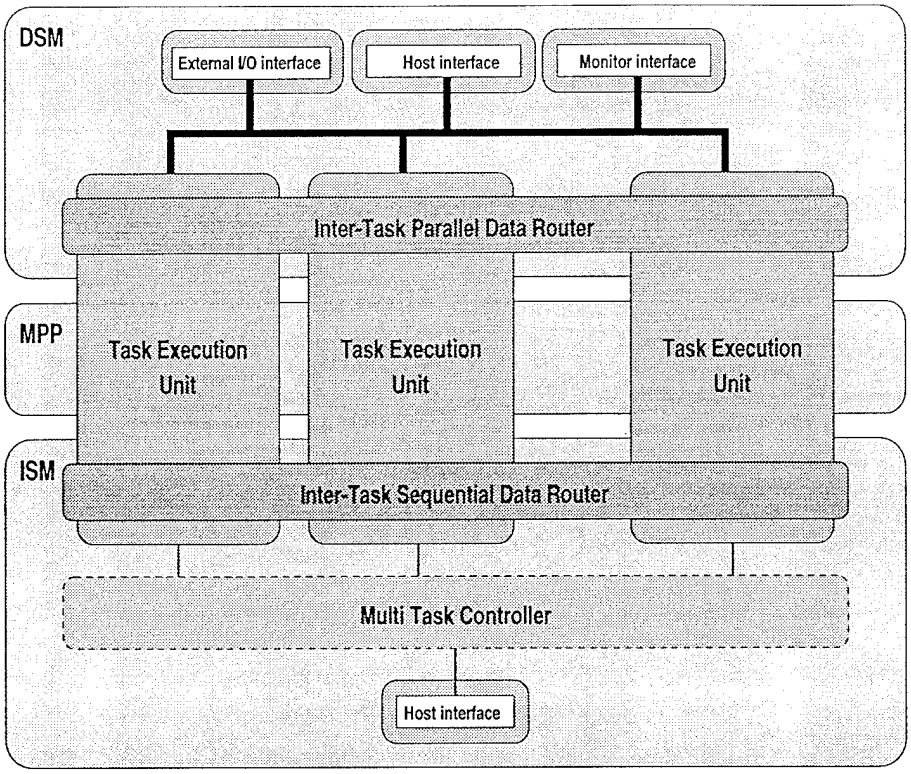

Figure 24: High-level Modular-MPC architecture

In section 2.3, it was observed that the natural parallelism of image processing applications is often characterised by a massive data-level parallelism (e.g. all pixels in a *sub-image* can be processed in parallel) and a modest control-level parallelism between *task-packages* or *tasks*. The common MPC architectures, MIMD and SIMD, introduced in section 2.4 exploit either control-level or data-level parallelism, but cannot make use of the respectively other level of parallelism. Therefore, a different architectural approach, known as Multiple Instruction control of Multiple SIMD (MIMSIMD), which can exploit massive data-level parallelism as well as the modest control-level parallelism has been employed for the Modular-MPC. Figure 24 depicts a MIMSIMD configuration of Task Execution Units (TEUs).

The "MIM part" of this configuration is implemented by a number of Task Execution Units (TEU), each executing *task packages* (see section 2.3). Depending on the control-level parallelism, the hierarchy of *tasks* in *task-packages* (see section 2.3) can either be executed sequentially in one Task Execution Unit or be spread over several TEUs. In the latter case, modest control-level parallelism is exploited by a number of TEUs working in parallel. However, the actual number of TEUs depends on a cost-effective compromise between temporal and spatial parallelism to achieve the minimum cost for given performance requirements. Finding this compromise requires balancing, similar to the balancing of the number of PEs vs the number of data channels shown in Figure 3.

In order to exploit the control-level parallelism, a multi-task controller generates a private control stream for each Task Execution Unit (TEU). Task Execution Units are connected via two routers. Parallel Data can be exchange via the Inter-Task Parallel Data Router (ITPDR, see also Appendix A.2.3.3), sequential data is exchanged via the Inter-Task Sequential Data Router (ITSDR).

Finally, TEUs can communicate with the outside via a number of interfaces. While sequential data can only be exchanged with the outside via the host interface which connects the Modular-MPC with the host workstation, a choice of interfaces is available for parallel data I/O:

- host interface

  In cases of moderate performance requirements for parallel data I/O, the transfer can be handled by the host workstation via the host-interface.

- external I/O interface

  For high-speed parallel data I/O and for specific data transfer protocols (e.g. HiPPI [Ame93], SCSI [Ame94]) a number of external I/O interfaces is provided.

- monitor interface

  The specific requirements of parallel data output to high-resolution displays is handled by a monitor interface, which provides all functionality necessary (e.g. frame store, D/A converter) to directly interface to a monitor.

While the control-level parallelism is exploited by several Task Execution Units (TEUs) working in parallel, the data-level parallelism is exploited within each TEU, each of which implements a SIMD structure. The remainder of this section is devoted to the detailed introduction of a TEU and its functional blocks.

A.2.1 Task Execution Unit (TEU) overview

As already mentioned above, the parallel processing in each TEU is based on the Single Instruction control of Multiple Data (SIMD) concept. Its implementation is based on ASP modules. Figure 25 shows a high-level view of the three main parts of a Task Execution Unit (TEU).

The **Massively Parallel Processor** implements a Parallel Process Execution Unit (PPEU) based on the Associative String Processor (ASP). The ASP is implemented as a string of ASP modules, emerging from research at Brunel University and developed by Aspex Microsystems Ltd. It was specifically designed for and has successfully demonstrated

- *machine versatility*, which is inherent in its architecture (see Appendix A.2.2)

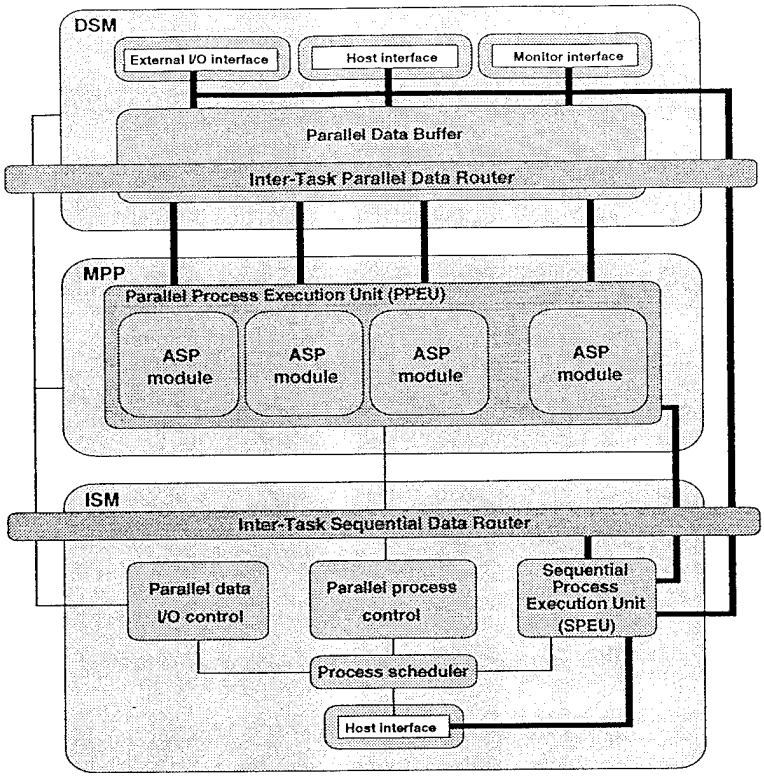

Figure 25: Modular-MPC Task Execution Unit (TEU)

- *performance scalability* due to the infinite scalability of the ASP string
- reduction in *size*, *weight*, *power* and *cost* due to the fact that the string topology of the ASP architecture has been specifically developed for the use of microelectronics (VLSI, WSI) and packaging (MCM) technologies
- operational efficiency, which is inherent in its architecture
- future proofing, through regular advances in microelectronics.

The Massively Parallel Processor (MPP) is presented in detail in Appendix A.2.2.

The **Data Stream Manager (DSM)** consists of

- interfaces for parallel data I/O
- Parallel Data Buffer (PDB)

  The PDB, which supports image patching, is crucial to minimise delays caused by parallel data I/O which manifest itself in $P_{oh}(0)$ (see section 2.4.1). The Parallel Data Buffer (PDB) provides storage for at least one, but usually several, images, similar to

a frame-store. Each Parallel Data Buffer (PDB) is connected to the PDBs of other Task Execution Units (TEUs) via the Inter-Task Parallel Data Router (ITPDR).

The Data Stream Manager (DSM) is connected to the Massively Parallel Processor (MPP) via a multi-channel, large bandwidth parallel data bus. The DSM and each of its functional blocks is discussed in detail in Appendix A.2.3.

The **Instruction Stream Manager (ISM)** consists of

- host interface
- process scheduler, which schedules
  - parallel data I/O control
  - parallel process control
    A single channel control connection joins the parallel process control with the MPP (SIMD concept).
  - Sequential Process Execution Unit (SPEU)
    This unit is provided to overlap sequential and parallel processing. Thus, it plays a crucial role in eliminating $P_{oh}(0)$ and $P_{oh}(i)$ for low i (by overlapping with parallel processing in the PPEU) as well as in the reduction of the times caused by $P_n(1)$ and $P_n(i)$ for low i (see section 2.4.1). The SPEU is implemented with a floating point processor and has a single channel sequential connection to the Massively Parallel Processor (MPP). Furthermore, a connection between the Parallel Data Store (PDS) and the SPEU provides a bridge between parallel and sequential data.

The Instruction Stream Manager (ISM) together with its functional blocks will be introduced in detail in Appendix A.2.4.

A.2.2 Massively Parallel Processor (MPP)

Figure 26 shows the overall structure of ASP modules in the context of a Task Execution Unit (TEU, see Appendix A.2.1). The MPP is implemented as an associative SIMD processor consisting of a string of (simple) Associative Processing Elements (APEs, see Appendix A.2.2.1). By implementing large numbers of APEs the need for patching can be significantly reduced. For example *tasks* and *processes* on a 1024×1024 image, executed on 65536 APEs require only 16 patches.

The Associative Processing Elements (APEs) are connected via the Inter-APE Communication Network (IACN). The IACN is a flexible network aimed at the navigation of data structures. It can be dynamically reconfigured, thus providing a cost-effective emulation of

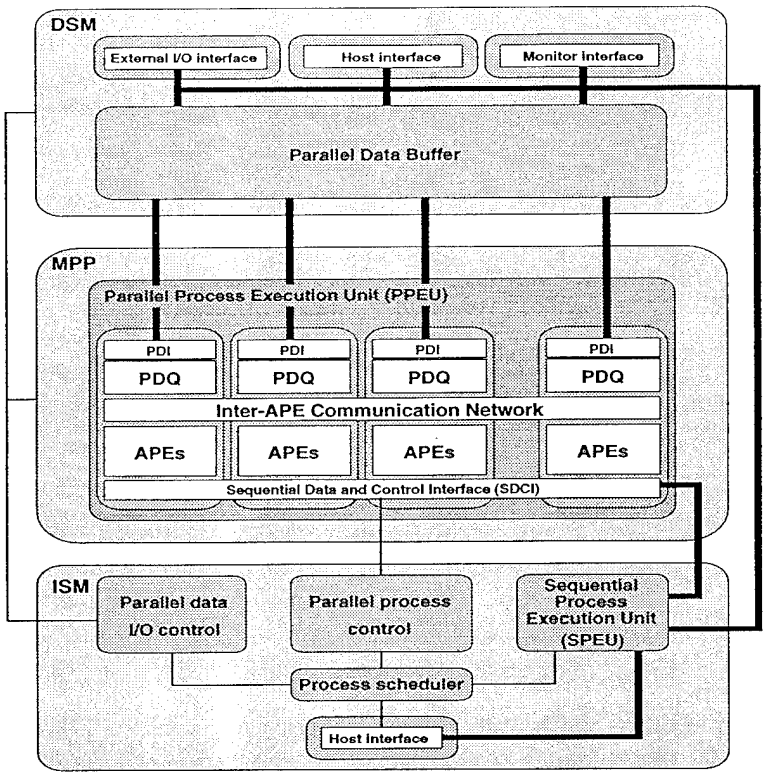

Figure 26: Modular-MPC ASP modules

common network topologies. As an activity-passing, rather than a data-passing, network it minimises data transfers. In fact, a different network dedicated to data reorganisation is provided in the DSM (see Appendix A.2.3.1.2). Consequently, time-consuming data transfers are only executed on the Inter-APE Communication Network (IACN), if they cannot be avoided otherwise.

The topology of the IACN is derived from a shift register and a chordal ring. The latter enables the IACN to be implemented as a hierarchy of ASP substrings. Thus, communication times are significantly reduced through automatic bypassing of those ASP substrings which do not include destination APEs. In a similar way, namely through bypassing of faulty ASP substrings, fault-tolerance of the ASP modules is guaranteed.

While being served with sequential data and control via the common Sequential Data and Control Interface (SDCI), each ASP module has a private Parallel Data Interface (PDI) for the transfer of parallel data. The Modular-MPC uses a hierarchy of parallel data pipelining to transfer parallel data form the interfaces to the outside (i.e. host interface, external I/O interface and monitor interface) to the Parallel Data Interfaces (PDIs) of the ASP modules and vice versa. The lowest level in this hierarchy comprises the Parallel Data Queue (PDQ). Data transfers between the PDQ and the ASP modules are called *Primary Data Transfers* (PDT). The PDQ is implemented with an orthogonal data queueing mechanism.

Data is loaded, overlapped with parallel processing, word-sequentially, bit parallel. It can subsequently be exchanged with the APEs in a word-parallel bit-sequential manner. Due to the massive bandwidth of this exchange (e.g. with a bandwidth of 40 Mbits/sec for each APE a Modular-MPC comprising 64k APEs has a bandwidth for Primary Data Transfers (PDTs) of 2.63 Tera bits/sec), the exchange time during which parallel processing has to be stopped, and consequently $P_{oh}(0)$ (see section 2.4.1), can be reduced to a minimum. In fact I/O overheads represented by $P_{oh}(0)$ due to this exchange are often well below 1 %. Note, that due to its orthogonal structure, the PDQ scales linearly with the number of APEs in the string.

ASP modules provide a simple means for scalability. Performance can be adjusted to the application requirements by changing the number of APEs per ASP module. I/O requirements, which manifest themselves in the required number of data channels, are met by changing the number of ASP modules.

The remainder of this description of the Massively Parallel Processor (MPP) is devoted to the description of an Associative Processing Element (APE). Further information on the APE architecture as well as the ASP modules can be found in [Lea88].

A.2.2.1 Associative Processing Element (APE) The Associative Processing Element (APE) is characterised by its simple structure. As shown in Figure 27 it consists of a 64-bit data register, a 6-bit activity register, a 70-bit comparator, a single-bit full adder, four status flags and control logic. The 6-bit set of activity bits is used to select subsets of APEs for processing.

The APE can operate in three different data modes. The 64 bit data register can be configured for

- storage and bit parallel processing of two 32-bit binary words
- storage and bit-parallel processing of four 8-bit ternary byte fields
- storage and bit-serial processing of one to three ternary contiguous bit fields of varying length (no more than 64 bits per field)

In order to adjust the size of the data register to the application requirements k physical APEs can be configured as a single virtual APE, providing a data register of $k \cdot 64$ bits.

The instruction set of the APE is based on 4 basic operations, match, add, read and write. These operations can be executed in two different modes, word-parallel bit-serial or word-parallel bit-parallel. (For a detailed discussion of the APE instruction set and processing modes see [Lea88]).

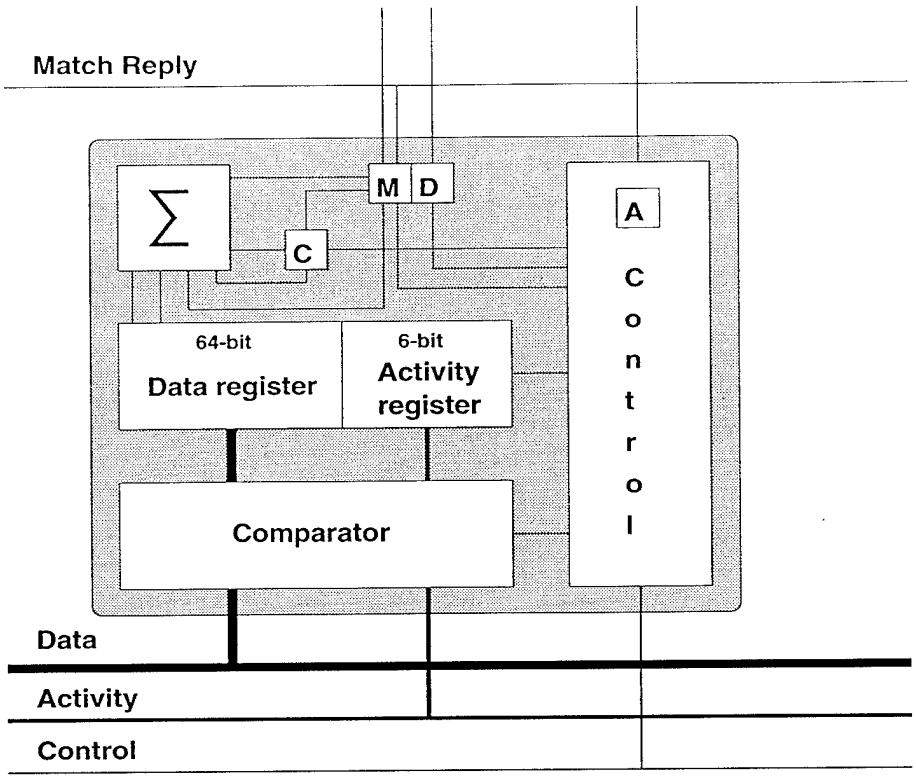

Figure 27: Associative Processing Element (APE)

A.2.3 Data Stream Manager (DSM)

Parallel processing applications vary considerably in their I/O requirements (see section 2). In order to achieve a high level of *machine versatility* (see section 2.1) the Data Stream Manager (DSM) is designed as a highly modular unit, which can adapt to the whole spectrum of I/O requirements. In particular, the DSM is designed to minimise I/O overheads (see section 2.4.1) through a successive increase in input bandwidth between the stages of a data pipeline and patching overheads (see section 2.4.2) by providing a mechanism for high-bandwidth patch exchanges.

Depending on the I/O requirements of a particular application, different data pipelines can be implemented.

- 1-stage pipeline

  In this configuration of the DSM the parallel data interfaces are directly connected to the ASP modules, parallel data is only buffered in the Primary Data Queue (PDQ, see Appendix A.2.2). Consequently any image storage is external. Two levels of parallel data transfers can be observed:

  – Secondary Data Transfer (SDT)

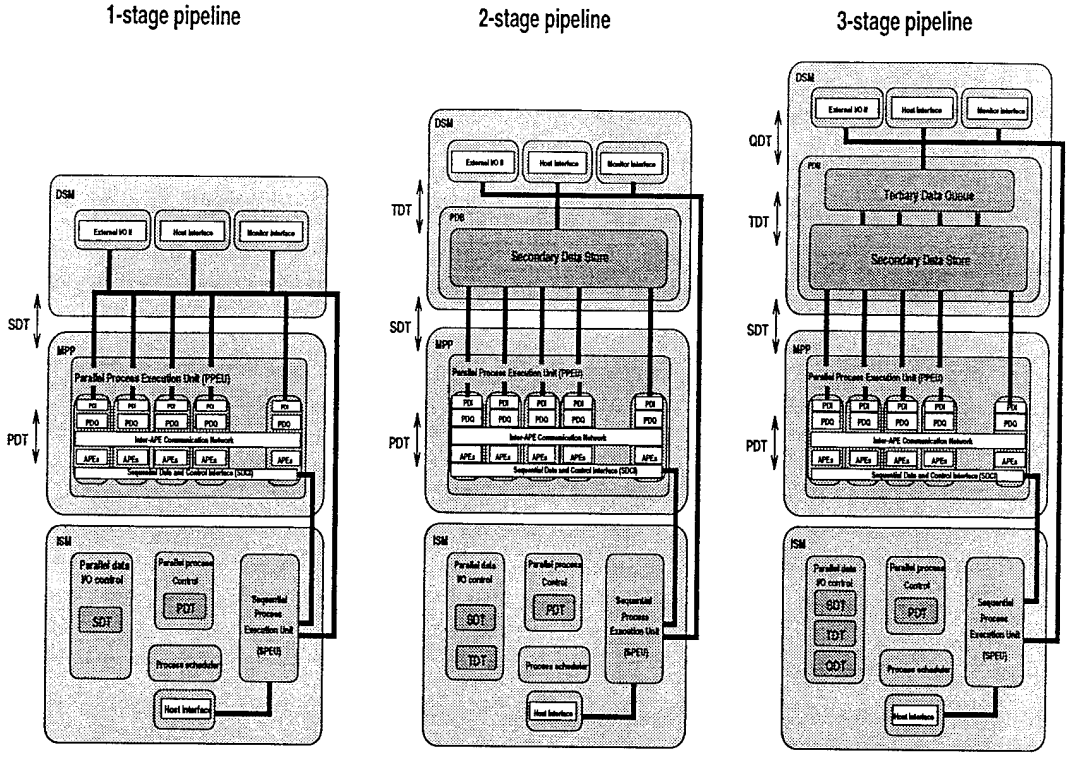

Figure 28: DSM configuration options

Data is transferred between the external interfaces and the Primary Data Queue (PDQ). This transfer is comparatively slow, since the PDQs of all ASP modules have to share a single connection to the interfaces. However, Secondary Data Transfers (SDT) can be overlapped with processing.

– Primary Data Transfer (PDT)

This word-parallel transfer between the Primary Data Queue (PDQ) and the ASP modules has already been introduced in Appendix A.2.2. It is a non-overlapped transfer with an extremely high I/O bandwidth.

- 2-stage pipeline

A 2-stage pipeline is configured by including the Secondary Data Store (SDS, see Appendix A.2.3.1) as a further stage in the parallel data I/O pipeline. In this configuration, patch processing is supported by the DSM. The Secondary Data Store (SDS) implements a multi-frame store, which is large enough to store at least one, but typically several, images. By storing or loading of sub-images while the processing of the current sub-image takes place, the inefficiencies caused by patching are minimised. Frequent exchanges of subimages with the MPP are made possible by a high bandwidth parallel data I/O. Compared to the 1-stage pipeline a further data transfer has been introduced:

- Tertiary Data Transfer (TDT)
  This transfer takes place between the external interfaces and the Secondary Data Store (SDS) via a single data channel for image transfer.
- Secondary Data Transfer (SDT)
  Other than for the 1-stage pipeline, this transfer now takes place between the Secondary Data Store (SDS) and the Primary Data Queue (PDQ). It implements a fast, multi-channel patch transfer which is overlapped with processing, thus minimising $P_{oh}(0)$ (see section 2.4.1). The number of data channels for the Secondary Data Transfer (SDT) is chosen to meet the application requirements.
- Primary Data Transfer (PDT) is implemented in the same way as described for the 1-stage pipeline.

- 3-stage pipeline

  With the introduction of a Tertiary Data Queue (TDQ, see Appendix A.2.3.2), the DSM can be configured as a 3-stage pipeline. Depending on the I/O requirements of the application and typical memories, a performance advantage can be achieved by including the Tertiary Data Queue (TDQ) in the parallel I/O data pipeline. Thus, it is used for the cost-effective minimisation of I/O overheads and as an intermediate storage to buffer data between the interfaces and the Secondary Data Store (SDS). Four levels of parallel data transfers can be observed for the 3-stage pipeline.

  - Quaternary Data Transfer (QDT)
    This transfer takes place between the Tertiary Data Queue (TDQ) and the interfaces via a single data channel.
  - Tertiary Data Transfer (TDT)
    In contrast to the 2-stage pipeline, for this configuration the TDT takes place between the Tertiary Data Queue (TDQ) and the Secondary Data Store (SDS). The number of connections for this transfer can be configured according to application needs.
  - Secondary Data Transfer (SDT) and Primary Data Transfer (PDT) are, compared to the 2-stage pipeline, unchanged.

The Data Stream Manager (DSM) does not only expose the high-level modularity which has been described so far. In fact, each stage of the different pipelines is designed as a modular unit.

A.2.3.1 Secondary Data Store (SDS) As already mentioned above, the Secondary Data Store (SDS) provides storage for one or more image frames for fast patch processing. Therefore, it needs to be scalable according to the storage requirements of the application, independently of the number of Associative Processing Elements (APEs) and independently of the number of I/O data channels for between the SDS and the MPP (see A.2.2).

Figure 29 shows the two modes in which the SDS can be configured.

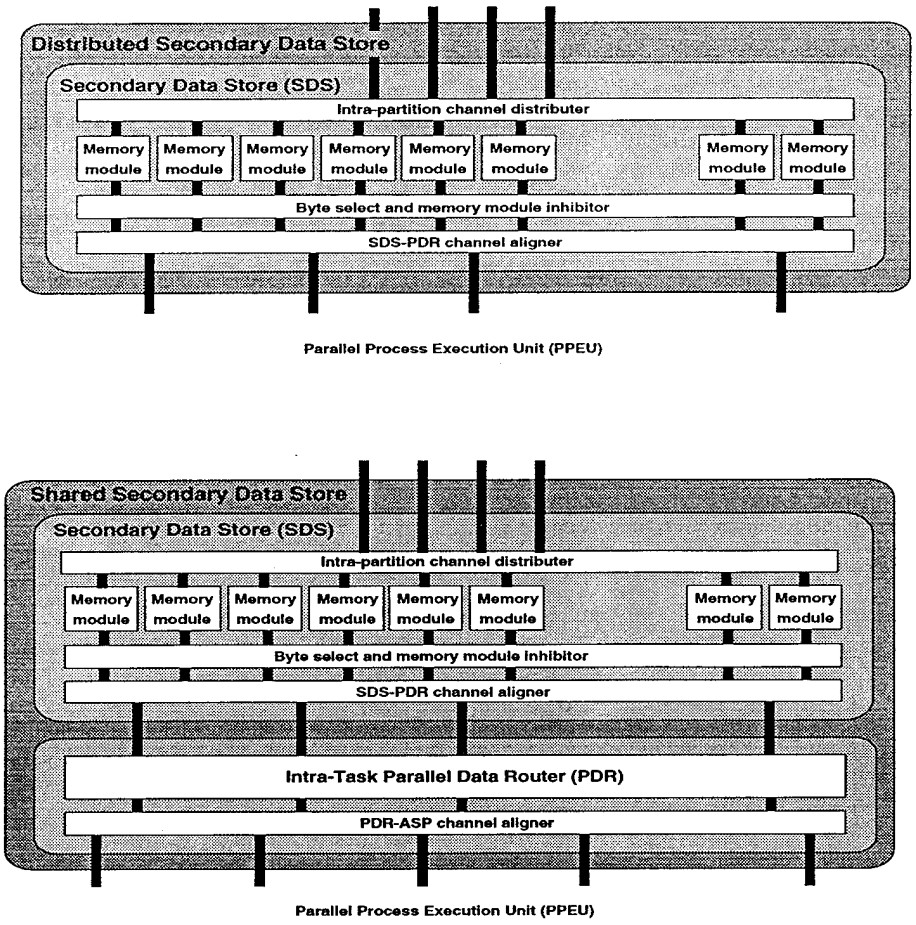

Figure 29: Distributed and global Secondary Data Store (SDS)

- distributed Secondary Data Store (SDS)

  This mode implements static routing between the different blocks of the SDS memory and the ASP modules. It is used for those applications where ASP modules access data only in assigned memory modules.

- shared Secondary Data Store (SDS)

  For a number of applications, ASP modules may access data in any memory module. In this case the SDS can be implemented in a shared mode with dynamic routing between memory modules and ASP modules. This dynamic routing is implemented with the Parallel Data Router (PDR, see Appendix A.2.3.1:2). Note that the PDR implements the preferred means for data reorganisation and complements the Inter-APE Communication Network (IACN) introduced in Appendix A.2.2. Consequently, through the flexible allocation of memory, the *application flexibility* of the Modular-MPC is increased significantly. A programmer sees the shared SDS as a single (shared) memory.

A.2.3.1.1 Secondary Data Store (SDS) memory The Secondary Data Store memory provides the storage for fast patching. It is implemented as a set of memory modules. Each module holds a fraction of the data kept in the SDS and is assigned its private I/O channels. The storage capacity of a memory module can be selected from a range of sizes.

Memory modules are grouped according to the current transfer

- for Tertiary Data Transfers (TDTs)

  Depending on the required bandwidth for TDT the set of memory modules is partitioned into subsets. The intra-partition channel distributer assigns each subset to one or more memory modules in the Tertiary Data Queue (TDQ, see Appendix A.2.3.2). For 2-stage pipelines the SDS appears as a single set of memory modules.

- for Secondary Data Transfers (SDTs)

  Depending on the required memory size per channel and the selected memory size for each memory module, one or more memory modules are assigned to an ASP module I/O channel. This assignment is implemented by the SDS-ASP channel aligner.

The SDS memory provides two modes of data filtering, implemented by the byte select and memory module inhibitor.

- byte select

  Any set of bytes in a storage word can be masked for write access such that only part of the memory word is updated. Byte masking is common to all memory modules.

- memory module inhibitor

  Write access to any memory module can be inhibited. The inhibition of a memory module can be data-dependent, i.e. only data with a specific signature is written to the SDS. The inhibition of one memory module is independent of all other memory modules.

A.2.3.1.2 Parallel Data Router (PDR) The Parallel Data Router (PDR) implements the dynamic routing required to implement a shared SDS. Therefore, it should not be seen as a general purpose network, but as a router. The PDR implements a cross-bar topology, i.e. any ASP module can exchange data with any memory module. The number of data channels in the PDR can be adjusted according to application needs.

The PDR is of crucial importance in order to cut a major source of $P_{oh}(0)$ (see section 2.4.1) due to I/O or communication. As opposed to other architectures, by reorganising data "on-the-fly" the time-consuming reorganisation via the network can be omitted. Data transfers via the PDR, and thus data reorganisation, can be fully overlapped with parallel processing.

A.2.3.2 Tertiary Data Queue (TDQ) Figure 30 shows the Tertiary Data Queue (TDQ) as part of a fully configured (3-stage) DSM. It provides intermediate buffering in order to achieve cost-performance benefits. This will become much clearer when seen from an implementation point of view (see section 4).

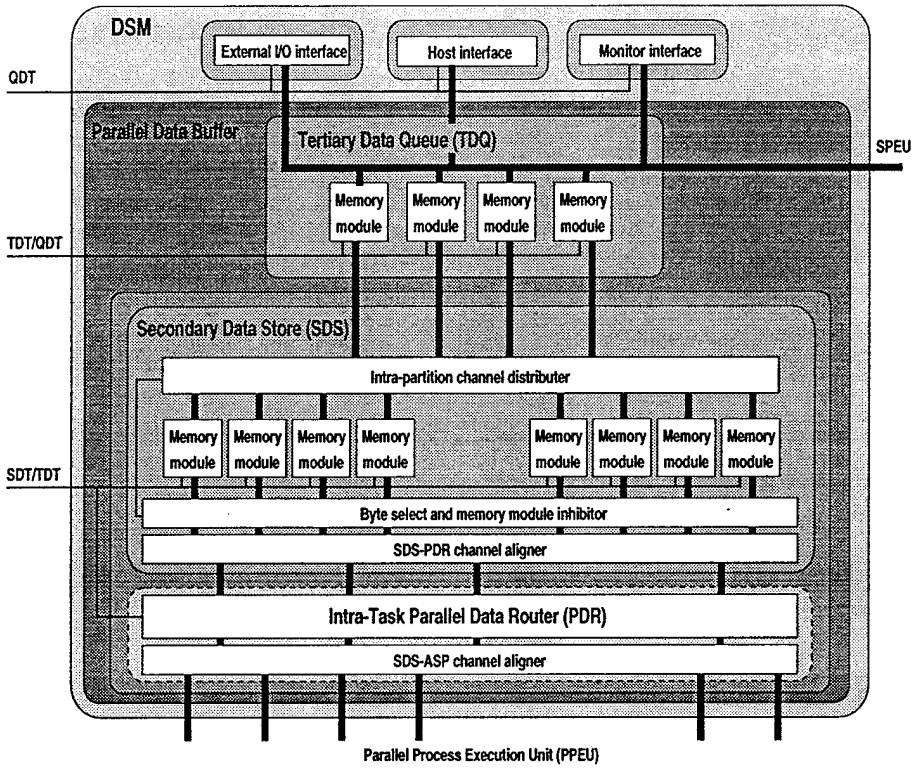

Figure 30: Tertiary Data Queue (TDQ)

Similar to the Secondary Data Store (SDS), the TDQ is implemented as a set of memory modules. As for the SDS, each memory module holds only a fraction of the data, has its private I/O channel and its capacity can be selected from a range of sizes.

The memory modules in the TDQ are grouped according to the executed transfer. For Quaternary Data Transfers (QDTs) all memory modules share a single data bus, the TDQ appears as a single block of memory. For Tertiary Data Transfers (TDT) one or more memory modules are assigned to a partition of memory modules in the SDS (see Appendix A.2.3.1). This assignment, which is implemented by the intra-partition channel distributer (cf Appendix A.2.3.1) depends on the required transfer speed.

A.2.3.3 Inter-Task Parallel Data Routing Having introduced the stages of the parallel data pipeline in the DSM, it is now possible to present the Inter-Task Parallel Data Router (ITPDR), introduced at the start of architectural description of the Modular-MPC

hardware, in more detail. As shown in Figure 31 there are three different possibilities to implement the ITPDR.

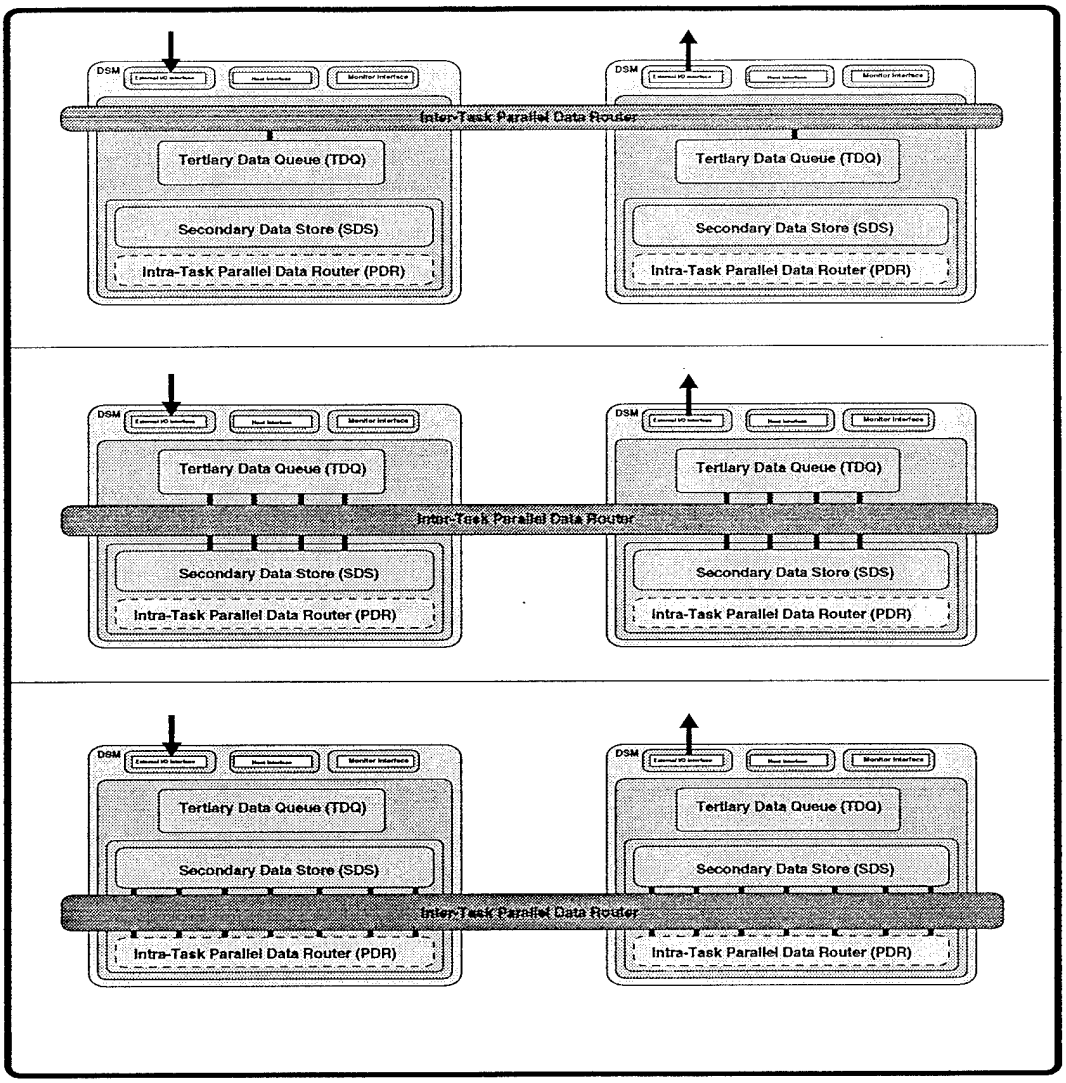

Figure 31: Inter-Task Parallel Data Router (ITPDR) configuration options

- ITPDR implementation between the interfaces and another stage of the data pipeline

  This implementation is the least complex of all possible implementations. However, the bandwidth which is provided for transfers via the ITPDR is relatively low.

- ITPDR implementation between Tertiary Data Queue (TDQ) and Secondary Data Store (SDS)

  This approach requires a higher implementation effort (i.e. higher hardware-complexity of the ITPDR) , which is rewarded with a higher bandwidth for inter-task parallel data transfers.

- ITPDR implementation between Secondary Data Store (SDS) and ASP modules

  This approach yields the highest bandwidth for transfers via the ITPDR, accordingly requiring highest implementation.

The choice of ITPDR implementation is governed by the application requirements. While some applications, especially those which have a massive data reduction between processing stages, will only require a small bandwidth for inter-task parallel data transfers, the performance of other applications will suffer significantly, if the bandwidth for transfers between SDS and ASP modules is not available for such transfers.

A.2.4 Instruction Stream Manager

Figure 32 presents an overview over the main functional blocks in the Instruction Steam Manager (ISM).

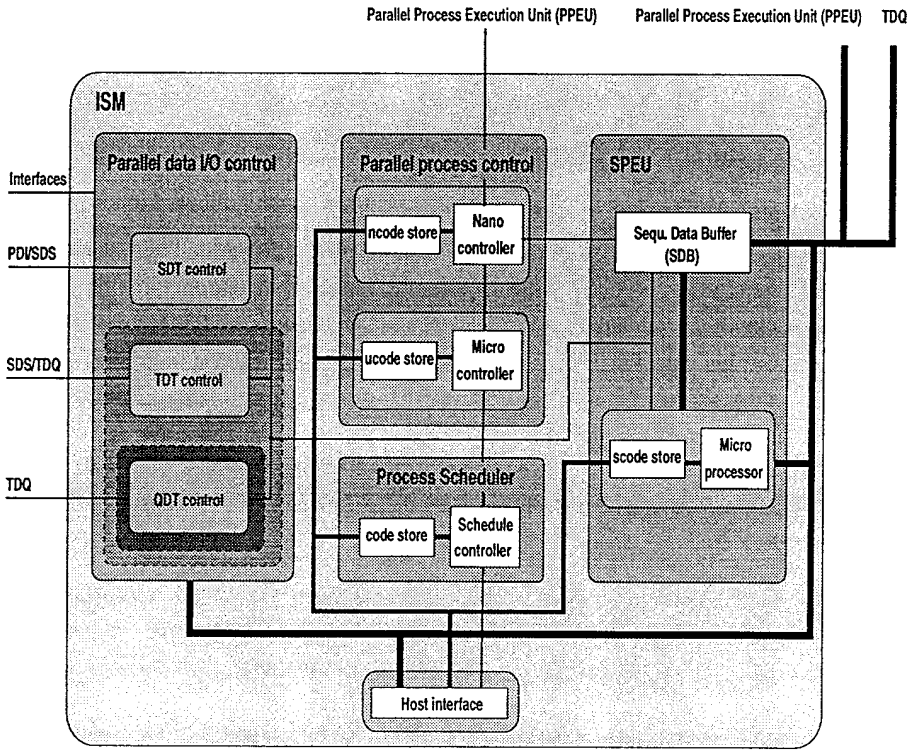

Figure 32: Instruction Stream Manager

A.2.4.1 Process scheduler The process scheduler controls and 'orchestrates' all other units in the ISM. Through remote calls it starts parallel processes in the parallel process

control, sequential processes in the Sequential Processes Execution Unit (SPEU) and it initiates data transfers controlled by the parallel data I/O control. Furthermore, the process scheduler uses the feedback from these units to align further scheduling of processes. As the main controlling unit in the ISM it is the only functional block which is directly connected to the host workstation via the host interface. The process scheduler is assigned a private code-store, which can be loaded via the host interface.

A.2.4.2 Parallel process control The parallel process control generates the control stream for the Massively Parallel Processor (MPP). As a cost-effective way to minimise $P_{oh}(0)$ (see section 2.4.1) due to control overheads, it is organised as a control hierarchy.

The *micro controller* calls blocks of operations which are remotely executed in the *nano controller*. While the nano controller, which is specifically designed for a fast repetition of blocks of operations, executes the remote call, enough time is created for the micro controller to organise the "housekeeping" (e.g. branching) of the parallel process control stream. Furthermore, the nano controller provides functionality to concatenate sequential data, which is stored in the Sequential Data Buffer (see Appendix A.2.4.3), and control to be sent as a single instruction to the MPP. Both, micro controller and nano controller have private code stores in which the required *task* and *processes* software library modules are downloaded via the host interface, prior to processing.

A.2.4.3 Sequential Process Execution Unit (SPEU) The Sequential Process Execution Unit (SPEU) is dedicated to the execution of sequential *processes*. Thus it plays a crucial role in minimising $P_{oh}(1)$ and $P_{oh}(i)$ for low i and in minimising the processing times for operations contributing to $P_n(1)$ and $P_n(i)$ for low i (see section 2.4.1). To this end, the SPEU includes a fast floating-point microprocessor with a dedicated code-store for sequential process code which can is downloaded via the host interface.

Furthermore, the SPEU includes a data store dedicated to sequential data, called Sequential Data Buffer (SDB). The SDB stores three kinds of sequential data:

- intermediate data used as sequential data in the MPP (see Appendix A.2.4.2)
- sequential data being evolved by the microprocessor of the SPEU
- parameters for the parallel data I/O control (see Appendix A.2.4.4)

Furthermore, the SDB acts as a bridge to move parallel Data from the Data Stream Manager (DSM, see Appendix A.2.3) to the Instruction Stream Manager (ISM) where it can be used as sequential data for further processing.

A.2.4.4 Parallel Data I/O control The parallel data I/O control governs data transfers between all levels in the Data Stream Manager (DSM, see Appendix A.2.3). Depending on the number of stages in the Parallel Data Store (see Appendix A.2.3), the parallel data I/O control implements a hierarchy of modules (see Figure 32):

- 1-stage pipeline

  only the Secondary Data Transfer (SDT) control is required

- 2-stage pipeline

  SDT control and TDT control have to be implemented

- 3-stage pipeline

  all levels of control, SDT control, TDT control and QDT control are required

The units which are controlled by a particular transfer-control depend on the number of stages in the parallel data I/O pipeline (see Figure 28). For example, in a 1-stage pipeline configuration the SDT control handles the transfers between the interfaces and the Primary Data Queue (PDQ, see Appendix A.2.2). However, in a 2-stage pipeline the SDT control handles the transfers between the Secondary Data Store (SDS, see Appendix A.2.3.1) and the PDQ. This fact is visualised by different line patterns representing the control paths for different DSM configurations in Figure 33, which shows a detailed view of the I/O control. All units surrounded by dashed lines are optional and will only be included for a particular configuration of the DSM.

Figure 33 demonstrates how the functional units in the parallel data I/O control can be partitioned in two sets:

- coordinating units

  These include SDT control, TDT control and QDT control. They coordinate transfers between units in the DSM by controlling address generating units.

- address generating units

  Although the rather large number of address generating units might seem confusing at first glance, these units follow a simple architectural principle.

  Each functional unit in the DSM parallel data I/O pipeline is assigned a private controller together with, if necessary, an address generator as follows:

  - the external interfaces are controlled by the interface control
  - the Tertiary Data Queue (TDQ, see Appendix A.2.3.2) is controlled by the TDQ address generator, which also provides the addresses for each memory module in the TDQ

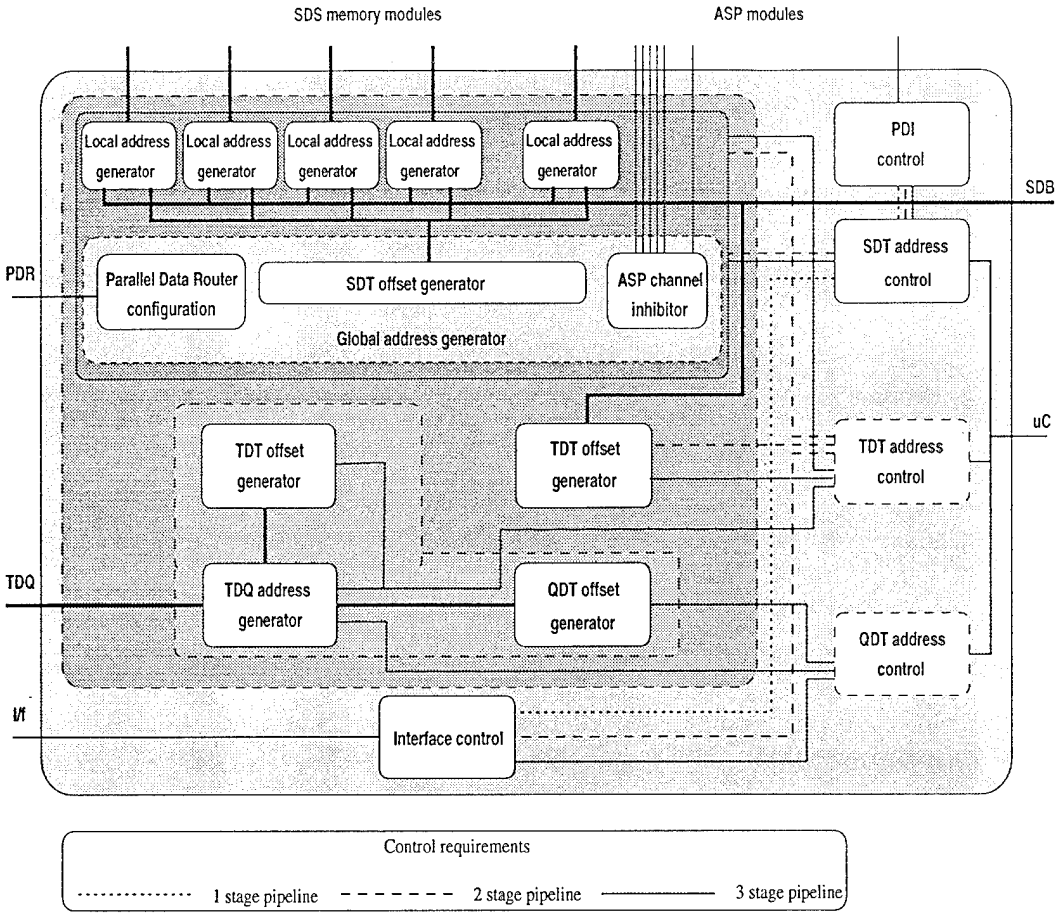

Figure 33: I/O control

- each memory module in the Secondary Data Store (SDS, see Appendix A.2.3.1) is controlled and provided with addresses by a local address generator
- the Parallel Data Router (PDR, see Appendix A.2.3.1.2) is configured and controlled by the PDR configuration unit
- the Parallel Data Interfaces (PDI, see Appendix A.2.2) are controlled by the PDI control and the ASP channel inhibitor

The Secondary Data Store (SDS) and the Tertiary Data Queue (TDQ) are accessed by two transfers, SDT and TDT, and TDT and QDT respectively. Therefore, their assigned address generators are able to switch between two different contexts, depending on the transfer which is executed at a given time.

For example, the TDQ address generators can receive parameters from the TDT offset generator (for TDT) and the QDT offset generator (for QDT). Parameters include an initial offset, an offset between consecutive addresses and the number of data elements which is to be transferred. New offsets can be calculated in the TDT and QDT offset

generators while the TDQ address generator sends the required addresses for the current block of transfers to the TDQ. Thus, by overlapping address generation and offset generation, any overheads due to address calculation can be minimised.

The offset generation for the local address generators is done in a similar way as the one described for the TDQ address generation. However, additional functionality is included for the case of a shared SDS (see Appendix A.2.3.1) when the Parallel Data Router is included in the SDS. In this configuration, for Secondary Data Transfers, offsets have to be generated for each local address generator, the parallel data router has to be configured and ASP channels have to be inhibited in cases of access contention to certain memory modules in the SDS. All this is handled by the global address generator, which executes these tasks overlapped with the actual address generation for SDT in the local address generators.

Note that parameters for the different address generators can also be downloaded from the Sequential Data Buffer (SDB, see Appendix A.2.4.3).

A.3 Software architecture

Figure 34 shows a high-level view of the Modular-MPC architecture and depicts of the different ways of accessing the Modular-MPC software. All users access Modular-MPC via a familiar programming environment and operating system.

Figure 34 is partitioned in two main regions separated by a bold line.

The region outside this line represents sequential programming environments used by the majority of users. Within the bold line, experts have direct access to Modular-MPC through the ASP programming language.

Sequential programming environments and their associated tools are industry standards (e.g. High-level programming languages, GUI, etc.) and as such are supported by software manufacturers. The user-environments are adapted to specific needs and knowledge of the user, who, depending on the chosen programming environment, requires varying knowledge of

- parallel programming
- Modular-MPC methodology
- Modular-MPC hardware and software

Parallel programming is based on software module library access. Control-level parallelism can be configured out of task libraries. Data-level parallelism is hidden in process libraries. Library modules consists of highly optimised ASP code, created by parallel programming experts with the ASP programming language (see Appendix A.3.2.1).

In the remainder of this section, both the sequential and the parallel programming environments are introduced in more detail.

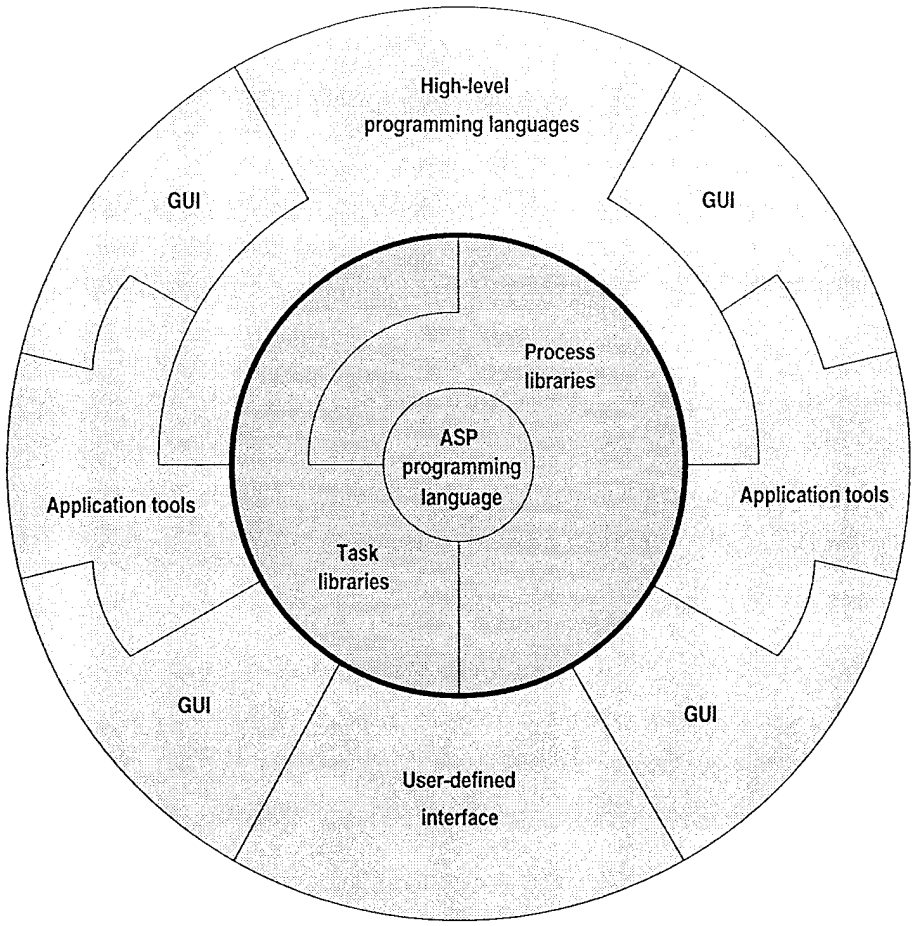

Figure 34: Modular-MPC software architecture

A.3.1 Sequential programming environment

As already mentioned above, the sequential programming environment is the environment most users would employ to access the Modular-MPC. This section presents the available interfaces, libraries and tools.

A.3.1.1 Interfaces Three different kinds of interfaces are provided, these being Graphical User Interfaces (GUIs), application tool interfaces and high-level language interfaces. However, as shown in Figure 34 users can also define their own interface, which can be tailor-made to their specific needs, to access the Modular-MPC libraries.

A.3.1.1.1 Graphical User interfaces (GUIs) Graphical User Interfaces (GUIs) are mainly used to access existing application packages. Using GUIs requires neither knowledge

of the Modular-MPC nor any knowledge of parallel programming. Potential users are end-users of an application, e.g. a surgeon who uses an application package for image guided surgery.

A.3.1.1.2 Application tool interfaces The purpose of these interfaces is to facilitate the use of application-tools (e.g. Photoshop [GS95], Analyze [RH90], IDL [Res94], IAP [HD92]). Application tools are provided as task libraries (see Appendix A.3.1.2). Using application tool interfaces requires no knowledge of parallel programming or the Modular-MPC.

Potential users include application developers who provide and maintain image processing application packages for end-users.

A.3.1.1.3 High-level language interfaces As all other interfaces introduced in this section, high-level language interfaces provide a sequential programming environment. They are based on standard languages (e.g. C, C++) and associated tools.

Due to their lower level of abstraction the usage of high-level language interfaces requires understanding of the Modular-MPC programming methodology. Parallel programming is based on the configuration of Modular-MPC task library modules (see Appendix A.3.1.2) using a high-level performance monitor (see Appendix A.3.1.3). Thus Modular-MPC software is transparently integrated, parallelism being incorporated in Modular-MPC process library modules (see Appendix A.3.1.2).

As shown in Figure 34 users of high-level language interfaces have two ways to obtain the software modules they require for their application:

- full-custom Modular-MPC task library modules

  If the required software modules are already part of a task library, they can be directly accessed. Otherwise, in order to incorporate the required *tasks* as software modules, these have to be generated by expert programmers (see Appendix A.3.2). They could, for example, be procured from Modular-MPC task library developers, who specialise in the development of such software modules.

  The advantage of this approach to programming via a high-level language interface is that the used software modules are always highly optimised. However, as already pointed out, they might not always be readily available.

- semi-custom Modular-MPC task library modules

  The second approach to acquire the required software modules, is to configure them from Modular-MPC process library modules. As discussed in Appendix A.3.1.2 process library modules provide a wide range of basic processes out of which any functionality on the task level can be configured. The configuration of process library modules is

supported by an optimisation tool (see Appendix A.3.1.3). However, the efficiency of software created via semi-custom Modular-MPC task library modules will normally be below the efficiency which can be achieved by incorporating full-custom task library modules. On the other end, this approach opens a path to "parallel programming" even for users who are not expert in the field and it provides a means for fast prototyping of applications.

Potential users of high-level language interfaces include application developers who provide and maintain image processing application packages for end-users and application tool developers who provide and maintain application-specific software tools

A.3.1.2 Libraries In the Modular-MPC software architecture libraries are the main means to encapsulate parallelism. They are highly optimised, created by parallel programming experts (see Appendix A.3.2). Modular-MPC library developers program in the ASP programming language (see Appendix A.3.2.1) which allows them to optimise ASP code on a low level of abstraction.

As shown in Figure 34 two levels of libraries are available:

- *task* libraries

  Task libraries are application-specific libraries which include a hierarchy of *task packages* and *tasks.*

  Examples for *task packages* in the area of image processing include volume-rendering and object detection. *Tasks* in this application area include median filter, sobel filter, Hough transform and convex hull.

- *process* libraries

  Process libraries provide basic operations. Thus they form a core of the Modular-MPC software. Examples for *process* libraries include

  - arithmetic library
    including among others add, subtract, multiply, divide
  - logic library
    including processes like AND, OR, NOR
  - statistics library
    including min-max search, match-and-count

A.3.1.3 Tools In the sequential programming environment four tools are provided:

- debugger

  The standard debugger provided with a programming environment can be used for Modular-MPC programming.

- performance monitor and profiler

  This tool gives the user high-level information about the performance, the efficiency and bottlenecks in the executed application.

- optimisation tool

  This tool is used for the generation of semi-custom task library modules (see A.3.1.1). It changes process library module interfaces to application specific interfaces, thus optimising the interaction between processes in a specific configuration of process library modules.

- configuration tool

  The configuration tool captures the rules governing the synthesis of a Modular-MPC configuration which matches the natural parallelism of a specific application (see Appendix A.1.1). In particular, it derives required a Modular-MPC configuration to meet application requirements cost-effectively. Hence, it can be used in the sequential programming environment in order to determine which Modular-MPC hardware configuration would be required for a given algorithm, thus optimising software development at an early stage.

A.3.2 Parallel programming environment

The parallel programming environment offers users direct programming access to the Modular-MPC. Therefore, a high-level of expertise in parallel programming as well as in-depth knowledge of Modular-MPC hardware and software is required. It is mainly used for library module development.

A.3.2.1 Interface In the parallel programming environment a single interface is provided, the ASP programming language. Parallel programming is based on the Modular-MPC hardware, the Modular-MPC system software and associative programming concepts.

The ASP programming language [Asp93] gives the programmer direct programming access to the Modular-MPC hardware thus enabling fine tuning of ASP code.

A.3.2.2 Libraries Programmers in the parallel programming environment can use library modules from the process libraries (see Appendix A.3.1.2), which provides basic operations and forms a core of the Modular-MPC software architecture.

A.3.2.3 Tools

- debugger

  The debugger for the ASP programming language provides a low-level view of the Modular-MPC hardware. The state of the main functional blocks in any Associative Processing Element (APE, see Appendix A.2.2.1) can be monitored.

- performance monitor and profiler

  The performance monitor and profiler for the parallel programming environment gives, other than the equivalent tool in the sequential programming environment (see Appendix A.3.1.3) a low level view of the performance of all major functional blocks in the Modular-MPC. It guides the user in the identification of bottlenecks which can subsequently resolve via the ASP programming language.

- optimisation tool

  In the parallel programming environment the optimisation tool, which has been introduced in Appendix A.3.1.3, is mainly used for fast prototyping and subsequent functional verification of new task library modules.

A.3.3 Operating System

Different operating systems are implemented on the different levels of the Modular-MPC control generation (see Appendix A.2.4), which are tailored to the functional requirements of each level.

- host

  The host runs its standard operating system (e.g.UNIX or VMS). In this environment, the Modular-MPC is controlled via remote process calls and gives feedback via interrupts.

- process scheduler

  The process scheduler (see Appendix A.2.4.1) runs its private OS to schedule the other functional units in the Instruction Stream Manager (see Appendix A.2.4).

- micro controller

  The micro controller (see Appendix A.2.4.2) runs a very reduced OS. It supports mainly sequencing and branching.

- parallel data I/O control

  The parallel data I/O control (see Appendix A.2.4.4) runs firmware which is downloaded at the start of the program. The required firmware depends on Modular-MPC configuration (in particular the configuration of the parallel data I/O control) and application I/O requirements.

- Sequential Process Execution Unit (SPEU)

  The SPEU (see Appendix A.2.4.3) runs a very reduced OS, which mainly supports I/O functions required for the microprocessor to access data in the SDB.

A.3.3.1 System initialisation System initialisation is achieved in five main steps:

1. downloading of libraries and programs into the different code stores (see Figure 32)
2. downloading of sequential data into the Sequential Data Buffer (SDB, see Appendix A.2.4.3)
3. initialisation of the process scheduler (see Appendix A.2.4.1) through the host workstation
4. initialisation of the other functional units in the ISM (see Appendix A.2.4), I/O control, SPEU and parallel process control

B Acronyms

APE	Associative Processing Element
ASP	Associative String Processor
ASTRA	ASP System Test-be for Research and Applications
AVIS	Advanced Vision Systems
DSM	Data Stream Manager
FPGA	Field Programmable Gate Array
GUI	Graphical User Interface
HDI	High Density Interconnect
IACM	Inter-APE Communications Network
ITPDR	Inter-Task Parallel Data Router
ITSDR	Inter-Task Sequential Data Router
ISM	Instruction Stream Manager
NEO	Near Earth Object Detection
MCM	Multi Chip Module
MIMD	Multiple Instruction control of Multiple Data
MIMSIMD	Multiple Instruction control of Multiple SIMD
MPC	Massively Parallel Computer
MPP	Massively Parallel Processor
PDI	Primary Data Interface
PDQ	Primary Data Queue
PDR	Parallel Data Router
PDT	Primary Data Transfer
PPEU	Parallel Process Execution Unit
SDCI	Sequential Data and Control Interface
SDS	Secondary Data Store
SDB	Scalar Data Buffer
SDT	Secondary Data Transfer
SIMD	Single Instruction control of Multiple Data
SIMM	Single In-line Memory Module
SPEU	Sequential Process Execution Unit
TDQ	Tertiary Data Queue
TDT	Tertiary Data Transfer
TEU	Task Execution Unit
VLSI	Very Large Scale Integration
WSI	Wafer Scale Integration