The VEGA Moderately Parallel MIMD, Moderately Parallel SIMD, Architecture for High Performance Array Signal Processing

Mikael Taveniku[2,3], Anders Åhlander[1,3], Magnus Jonsson[1] and Bertil Svensson[1,2]

1. Centre for Computer Systems Architecture, Halmstad University,
Box 823, S-301 18 Halmstad, Sweden

2. Department of Computer Engineering, Chalmers University of Technology,
S-412 96 Göteborg, Sweden

3. Ericsson Microwave Systems AB,
S-431 84 Mölndal, Sweden

Abstract

In array radar signal processing applications, the processing demands range from tens of GFLOPS to several TFLOPS. To address this, as well as the, size and power dissipation issues, a special purpose “array signal processing” architecture is proposed. We argue that a combined MIMD-SIMD system can give flexibility, scalability, and programmability as well as high computing density. The MIMD system level, where SIMD modules are interconnected by a fiber-optic real-time network, provides the high level flexibility while the SIMD module level provides the compute density. In this paper we evaluate different design alternatives and show how the VEGA architecture was derived. By examining the applications and the algorithms used, the SIMD mesh processor is found be sufficient. However, the smaller the meshes are the better is the flexibility and efficiency. Then, based on prototype VLSI implementations and on instruction statistics, we find that a relatively large pipelined processing element maximises the performance per area. It is thereby concluded that the small SIMD mesh processor array with powerful processing elements is the best choice. These observations are further exploited in the design of the single-chip SIMD processor array to be included in the MIMD-style overall system. The system scales from 6.4 GFLOPS to several TFLOPS peak performance.

1. Introduction

Signal processing applications in the area of sensor array processing place computing demands on the processor in the order of from tens of GFLOPS to several TFLOPS. In addition, there are latency demands and constraints on power dissipation and size depending on the type of application. For example, an airborne fighter radar has tight size and power dissipation requirements, while a ground based surveillance system has not.

Based on radar application examples described in earlier publications [1][2] and on methods described in the literature on array signal processing the computations used in these applications have been identified. These are to the largest extent FIR filters, FFTs, QR-decompositions, SV-decompositions, matrix by matrix multiplications, and other similar calculations. Furthermore, the data is inherently parallel since there are several receiver channels in a phased array antenna system and the data from each channel is divided into range samples. Each of these samples belongs to a pulse, which in turn belongs to a pulse group. This gives the application and its algorithms the following properties:

1. The algorithms are matrix oriented, i.e., data can be divided into matrices which often can be processed independent of each other.
2. The data in the matrices can often be processed by independent rows or columns.
3. The processing can be split into “channels” with identical, independent processing.
4. The processing in each channel can be divided into functional blocks which, in turn, can be computed in parallel (in a pipelined fashion).

These properties make the applications naturally lend themselves to systolic array implementations, as shown by several authors (e.g., [3] and [4]). Using systolic arrays means that generality is traded for increased performance. In most cases the processor system will be specialized towards one specific data size and type of algorithm. In some applications this might not be a problem. However, it is a problem for a multipurpose radar where both matrix sizes and algorithms change during operation. At the other end of the architectural scale, there exist commercial distribut-

ed memory MIMD (Multiple Instruction streams, Multiple Data streams) systems, such as the Mercury RACE [5], which are highly flexible but they can not provide enough computational density. Therefore, we have developed a system in between the pure systolic array and the distributed memory MIMD system. It is a system which is specialized toward array signal processing and matrix calculations. It can cope with changes in matrix sizes and algorithms, and at the same time it is scalable in performance. To achieve this, as much generality as possible is kept, while not sacrificing performance.

The path taken is to use the SIMD (Single Instruction stream, Multiple Data streams) concept in the compute engines in combination with a modular MIMD system architecture. Thus, the closely coupled SIMD model is used to maximize performance and still have programmability on module level, while generality on a higher level is maximized by using the MIMD model as system architecture. An overall view of the system, which we call VEGA, is shown in Figure 1.

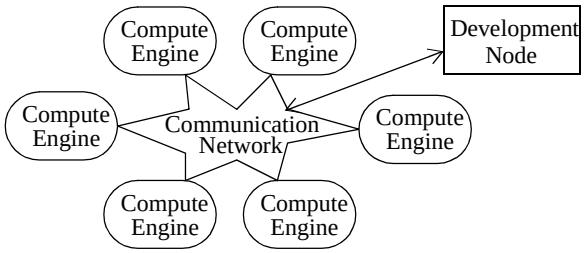

Figure 1: The VEGA Modular system. The compute engines comprise small SIMD modules.

The compute engines, see Figure 2, act as stand-alone computers. They consist of three parts: A network interface for communication with other engines, a node controller, and a data processor.

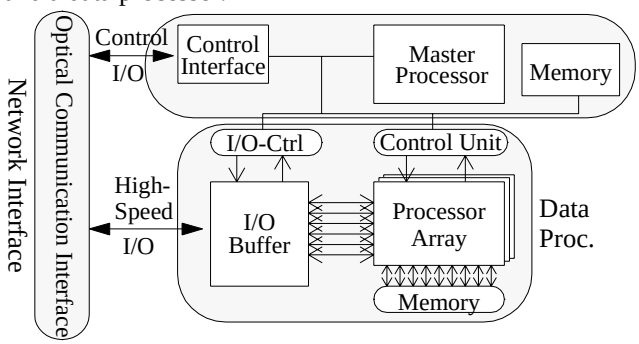

Figure 2: A Compute Engine.

The data processor is an SIMD processor, consisting of a control unit (CU) for processor and I/O control, input and output buffers, memory for coefficients and partial results, and the array of processing elements (PEs). In the following sections we consider the design of the VEGA system. The design comprises the internal organisation of the compute engines, the PEs, and the intermodule communication network. Finally we discuss some system issues.

2. Compute engines

The design of the compute engines and their PEs has been driven by the following requirements: A module shall be able to efficiently perform all steps of the signal processing chain, make it possible to handle variations in the problem size, have a low latency, be programmable, and implementable with today's state of the art technology.

From the work on algorithm mapping and the analysis of the sample systems, in combination with extrapolation to future larger systems, it is clear that it is advantageous to have as high performance in the modules as possible while maintaining programmability and flexibility. This, on one side, suggests large PE arrays for high performance, but on the other side single PEs for flexibility. The number of nodes must be kept low, a couple of tens, in order to make the programming of the MIMD system manageable. The easiest way to achieve this is to make the nodes powerful enough to compute a full function or pipeline stage and thereby make it possible to utilize parallelism on the MIMD system. Considering this, and the demands given by the applications, a node should have a performance in the order of 5 to 50 GFLOPS in order to meet the requirements and to allow the system to scale to one TFLOPS. Also latency requirements must be considered.

In our earlier work, two different approaches for the module architecture have been investigated. These are: A moderately large mesh array [1] and an array of meshes architecture [2]. We briefly review these and add a third alternative: a single-chip small processor array.

2.1 A data-sized mesh

The module in the first approach consists of 1024 PEs arranged in a 32x32 mesh configuration with row and column broadcast lines, see Figure 3.

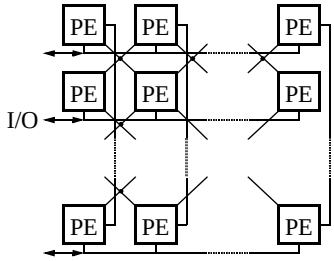

Figure 3: A 32-by-32 SIMD mesh processor array.

This processor array was suggested for a 32-channel multiple-beam airborne application with MVDR beamforming. The 2D array has the same size as the matrices in the data set. When the algorithms were mapped onto this array it was shown that the mesh was a suitable architecture, the algorithm mappings were straightforward, simple,

and fairly efficient. However, we found that, in certain parts of the calculations, the efficiency was rather low because of limited potential for parallelization of the calculations.

2.2 An array of meshes

The array of meshes module was designed to, in some extent, overcome the problems with processor utilization and those associated with large arrays. It consists of relatively small PE arrays, where the PEs are arranged in the same manner as in the previous machine. However, the arrays can in turn be arranged as needed by the application. These modules were evaluated in an example of a ground based system with 64 receiver and signal processing channels [2]. For this system the processor modules were arranged in eight 8-by-8 PE modules, see Figure 4. By using this arrangement, i.e., an array of meshes processor, it was possible to achieve better processor utilisation than in the previous machine. In general, the algorithms need only to be parallelized onto the small 64 processor array. In fact, if there are several identical calculations, a parallelism degree of 8 in each is enough. From the application it follows that it is also relatively easy to parallelize almost any signal processing algorithm onto this architecture and thereby this machine can gain some programmability and generality.

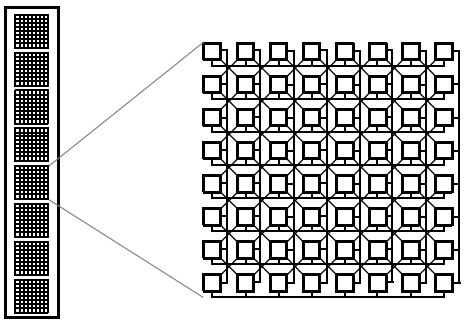

Figure 4: An 8 by 8-by-8 array processor.

This solution shows both good performance and scalability. However, if the PEs are to be full 32-bit floating point processors, it is, today, difficult to fit 64 PEs and a control unit on a single chip.

2.3 A single-chip mesh

We note that, in general, the algorithm mappings get more efficient when the processor mesh gets smaller. An important reason for this is that a small problem (i.e., a problem with a limited degree of parallelism) needing a fast solution is hard to map efficiently onto a large array.

Based on the VLSI implementation results that will be presented later in this paper, we estimate that a 16 PE array would be a reasonable choice for a single-chip solution. The array organization is shown in Figure 5.

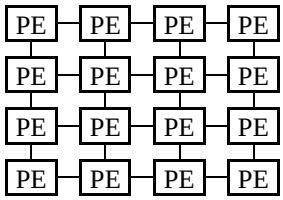

Figure 5: A 4-by-4 PE single-chip processor array.

This processor array is different from the previous ones in that it is not using broadcast lines and that it has a four-neighbours interconnect instead of the X-grid used in the previous machines. The longest path in the processor becomes six steps instead of two as in the larger array with broadcast lines. However, by using local interconnect only, the clock frequency of the interconnection network can be increased. Also, due to folding effects, the row and column broadcast operations are only 1/4 as frequent as in the 8-by-8 mesh. A broadcast operation is here emulated by a series of nearest neighbour communications which are overlapped with the calculations.

This processor array also avoids the problems with instruction broadcast and processor synchronization that we find in larger arrays. The single-chip solution also avoids problems caused by process variations between chips. This means that we can increase the operating frequency of the processors to be in parity with modern microprocessors.

In conclusion we note that the mesh communication network is sufficient for the algorithms considered and that the single chip processor module is a promising way of achieving high computing density. Furthermore the flexibility and programmability aspects are good for this array.

3. Processing element design

The architecture of the individual PE is of course an important part of the processor array design. Again, three different approaches are compared: Low complexity, high clock rate, bit-serial PEs; few-bit parallel PEs; and fully parallel PEs. Actually, the last two types of PEs were derived when the problem was formulated to minimise the delay-area product for the PEs. Instruction statistics from the applications were used, but no application specific parallelism constraints were assumed. A requirement in this work is that the computing is done on 32-bit floating point numbers.

3.1 A bit-serial PE

The first design builds on the idea that a bit-serial processor can be a powerful and efficient PE for a processor array, especially since a vast number of these small PEs can fit on a single chip. The design builds upon other work in the group [6] that show that such PEs can be highly computationally efficient and have low power consumption.

Since the computing is done on 32-bit floating point numbers, a bit-serial floating point unit was constructed. This unit is shown in Figure 6. It consists of a 24-bit bit-serial multiplier, three shift registers for mantissas, and an exponent unit (8-bit parallel).

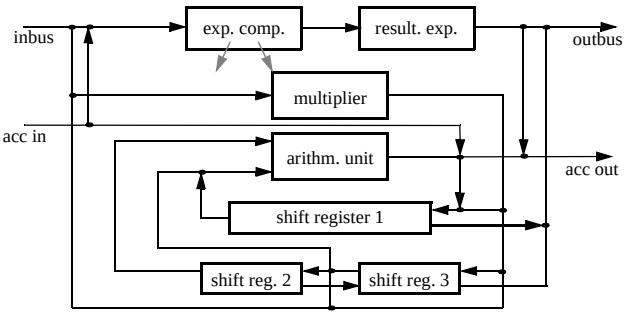

Figure 6: A bit-serial floating point PE.

The size of the processor multiplier is about 1000 gates and the whole ALU comprises 3000 gates. This PE is optimized for processing multiply, add and multiply-and-accumulate operations. The peak performance of the PE is 0.03 FLOPS/Hz for multiply-and-accumulate operations. Since the computational power per PE is relatively low, many PEs are required to fulfil the performance demand for a processor module. It should be possible to implement, say, 1024 PEs on a chip, if no register file is needed.

3.2 Bit-parallel processors

The bit-parallel processor structures were inspired by studies of computer arithmetic units in modern microprocessors. These have high performance arithmetic units, but only a few of them. Furthermore, in the processors a great effort is made to extract instruction level parallelism and to hide memory access times, thus making the processors, to a considerable extent, consist of cache units, branch prediction units, memory controllers, etc. These features are necessary if sequential multiprocessing with non-predictable execution is assumed, as in general purpose computing. However, for signal processing they are not necessary.

The idea for this PE architecture is to combine high ALU performance and single-chip implementation as in modern processors, but without unnecessary units. Furthermore, a data-parallel programming model, and predictable program execution, are used. With the objective of maximizing the performance per unit area, we made implementation studies of a spectrum of designs. These implementations have been done in one 0.6µm three metal layer and one 0.35µm five metal layer CMOS process, both developed by LSI Logic, Inc. From these implementations the PE shown in Figure 7 was derived. The PE consists of a communication controller, communication registers, a register file, and a pipelined bit-parallel floating point ALU. The register file is 32 words deep and has four ports.

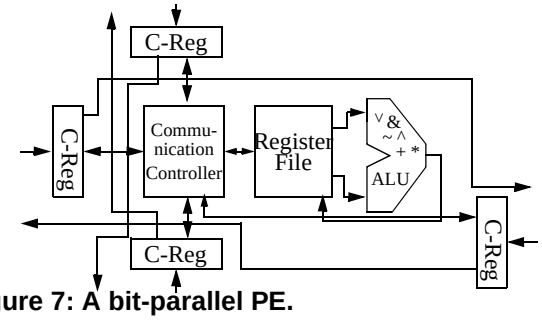

Figure 7: A bit-parallel PE.

The ALU is a combined multiply and add pipeline, which makes the design fast and compact. Several different ALU variations have been evaluated [7]. The integer multiplier and adder designs have first been implemented and evaluated against the delay-area product (D*A), i.e., the delay for an operation multiplied with the area of the unit. The result is shown in Table 1. For the adder designs, the delay is measured for a 24-bit addition. In the multiplier case, the delay is measured for a 24x24 bit multiplication. The table also shows combinations of multipliers and adders. For these combined elements, a 50/50 distribution of additions and multiplications is assumed. D*A is normalized against the bit-serial implementations. Note that the clock frequencies of the combined units are determined by the slowest of their integral parts.

Adder Type	Area (gates)	Delay 0.6µm	D*A 0.6µm	Delay 0.35µm	D*A 0.35µm
Bit-serial	40	21.6ns	1.0	16.8ns	1.0
Carry select	930	3.2ns	3.9	1.8ns	3.5
CLA 6 x 4-bit	760	4.3ns	4.3	2.6ns	3.9
CLA 4 x 6-bit	970	4.3ns	5.5	2.5ns	4.8
Carry ripple	360	13.7ns	6.5	7.9ns	6.2
Pyramid 3 x 8-bit	600	6.2ns	5.0	4.0ns	4.5
Multiplier Type					
Bit-serial	1200	174ns	1.0	79ns	1.0
Iterative	1800	62.4ns	0.53	27.4ns	0.50
Binary tree	10200	10.3ns	0.51	5.55ns	0.86
Pipelined bit-serial	1200	-	-	52ns	0.66
Pipelined 3-stage	17200	-	-	2.5ns	0.42
Multiplier-Adder Combination		Area (gates)	Delay 0.35µm		D*A 0.35µm
Bit-serial mult / Bit-serial add		1340	54ns		1.0
Bit-serial mult / CLA 6 x 4-bit		1417	56ns		1.1
Iterative mult / CLA 6 x 4-bit		1992	21ns		0.59
Binary tree mult / Carry select		15969	5.6ns		1.24
3-stage tree mult / Carry select		17313	2.5ns		0.59

Table 1: Results of implementation studies. CLA = Carry Lookahead Adder.

The bit-serial adder has the best D*A, while the iterative multiplier is best among the multipliers. When the units are combined, the D*A is smallest for the iterative multiplier combined with the 4-bit CLA, and for the pipelined multiplier with the CSA. Thanks to optimizations, the combined units comprise fewer gates than the sum of gates in their constituent parts.

When floating point support is added, the area increase for the parallel units is approximately 90%, while the area for the bit-serial processor will increase with 170%. For the iterative multipliers and adders the increase is somewhere in between. From this we conclude that the pipelined binary tree multiplier, combined with a CSA, is the most suitable combination. The implementation of this ALU used approximately 22000 gates excluding pipeline registers and register file. The maximum delay is 2.1ns for the blocks and the cycle time is 2.5 ns.

To complete the PE, a 32 word register file, pipeline registers, and control logic add another 15000-20000 gates. The size of 16 PEs, see Figure 5, would then be about 650 000 gates. This single-chip SIMD microprocessor would have a peak performance of 6.4 GFLOPS which is within the range of the requirement for the modules.

3.3 Discussion

Three interesting observations can be made based on the implementation studies. First, when moved from the 0.6µ technology to the 0.35µ technology, the complex designs gained more than the others in D*A. Second, the optimization options available to speed up the bit-serial design, such as dynamic latches and logic, also, with good efficiency, apply to the other designs. Third, algorithmic optimizations, such as Booth encoding for the multiplier, can be applied to the bit-serial processor, but it is hard to gain performance by them. It can be concluded that the more complex the design is, the more it can gain from algorithmic optimization and denser implementation technology.

In the algorithm studies [8] we found that the most frequent operations were multiplication and addition. Furthermore, 50% of the operations could be organized as multiply-and-accumulate operations. Implementations, with ideas from Quach and Flynn [9], have shown that adding multiply-and-accumulate capability increase the ALU size with 30-40%. This would improve the D*A with 15% for the system. It was, however, difficult to keep the same cycle time as before since the critical path in the normalisation stage became longer. It is therefore not certain that this will increase the efficiency that much. Therefore, we have left it out in the first design.

A possible implementation of a division unit has also been considered. Although the division and inverse square root operations occur only once in a thousand operations, they are in the critical path of the QRD, the SVD, and the back substitution algorithms [10]. Therefore, when the size of the processor array grows relative to the problem size, it will become more and more worthwhile to implement such a unit, due to the fact that more and more PEs will be idle waiting for the division to be completed.

A stand-alone division unit similar to the one in the MIPS R10000 would cost approximately 20000 gates and thus seriously affect the D*A. However, if multiplicative methods [11], [12] are used it is possible to implement, with small hardware modifications, both square root and division with a combination of software and hardware. This would make the PE marginally larger but the division would only require 9 operations.

We also compared the bit-serial and the bit-parallel designs with each other. Within the same chip area the bit-parallel processor reached at least four times the performance of the bit-serial one. Thus, we have found an efficient PE in the D*A sense.

4. System design issues

The final system is a scalable moderately parallel MIMD system with moderately parallel SIMD modules interconnected with a scalable, deterministic optical interconnection network. The applications are implemented on the system as follows. A multi-mode radar application can in general be described as a set of independent modes of operation. Furthermore, each mode is described as a series of transformations on the sampled data stream. In addition to this there are control functions and registration functions controlling and monitoring the system.

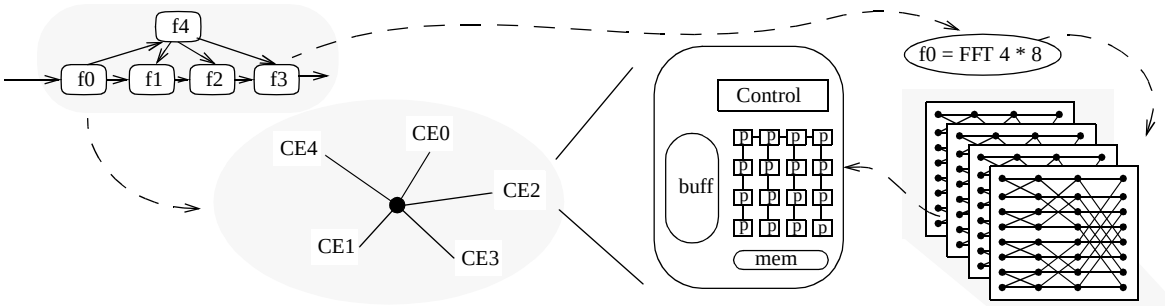

Figure 8: The MIMD system enables a functional decomposition and allocation of the program to the modules. Each function or block of functions is then mapped onto the processors in the module.

The system implementation process is illustrated in Figure 8. First, the modes of operation can be designed separately. The modes are then functionally decomposed and laid out on the system (MIMD-level). In this stage only the computational demands and the network bandwidth demands need to be considered. When the functions are assigned to processing nodes these can be further mapped onto the processors and buffer space can be allocated. This task is simplified by the fact that there is only a limited number of algorithms used, and therefore a function library can be used and developed. Furthermore, by the properties of the application, data is inherently parallel and is therefore easy to map onto these small processor arrays.

The chosen approach gives: (i) intuitive use of the MIMD system, (ii) understandable high level functional decomposition, (iii) scalability on MIMD level, and (iv) efficient mapping of algorithms because of small modules.

As an example we show how the signal processing in a ground based surveillance radar system is implemented using this approach [2]. In the system, 64 lobes are created using 64 receiver channels. The data flow in one mode of operation can be described as shown in Figure 9.

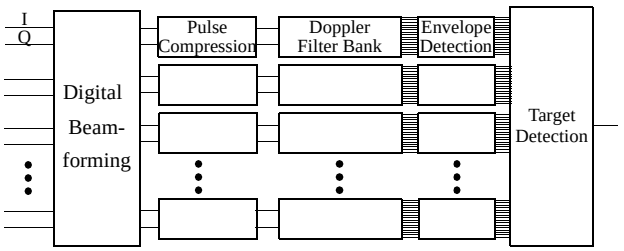

Figure 9: Description of one mode of operation in the 64-channel ground based radar system.

The total computational demand is 40 GFLOPS. The nodes in this system are of the array-of-meshes type with a sustained performance of 4 GFLOPS. The functions are mapped onto the nodes as shown in Figure 10. The buffer memory modules are used for corner turning of data. The algorithms are then individually mapped onto the specific processor array, in this case an 8-by-8-by-8 mesh processor as shown in Figure 4.

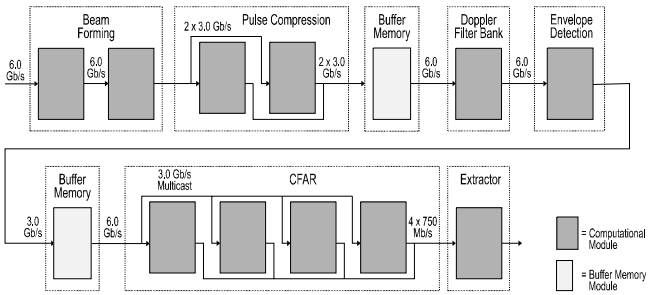

Figure 10: Allocation of link bandwidth and functions to the processing nodes.

5. Communication network

Although control I/O and high-speed data I/O are logically separated in the module (see Figure 2), a single network interface handles all inter-module communication. Since each module itself can have a sustained output data rate of several Gb/s, we need a powerful interconnection network. Another important feature of the network is the ability to guarantee that the time constraints are met.

We suggest two different ways of implementing the communication network. Both employ fiber-optic technology and support guaranteed timely delivery. The first network is a WDM (Wavelength Division Multiplexing) star network which scales to large systems, and the second is a fiber-ribbon ring network suitable for low-end systems.

5.1 WDM star network

For the star network a medium access protocol called TD-TWDMA (Time-Deterministic Time and Wavelength Division Multiple Access) and real-time services to guarantee that packets arrive before their deadlines have been developed [13]. The WDM technique gives the opportunity to simultaneously have multiple wavelengths (hereafter called channels), each carrying an information stream of several Gb/s. Commercial WDM systems for telecommunication exist and we can foresee cheaper short-distance WDM systems to appear.

By grouping the modules into clusters we can keep much of the traffic inside the clusters. Each cluster employs the WDM star configuration in which a passive optical star broadcasts transmitted data on all optical wavelength channels to all other nodes in the cluster (Figure 11). Two fibers, one for each direction, connect each node in the cluster to the star. Each node transmits on a specific channel but can tune in any channel for reception. In this way, broadcast transmissions are possible.

Large systems employ the star-of-stars configuration, instead of using a single cluster. All clusters are connected to a backbone cluster via electronic gateway-nodes to obtain wavelength reuse. The network scales up to systems with several hundreds of nodes [13].

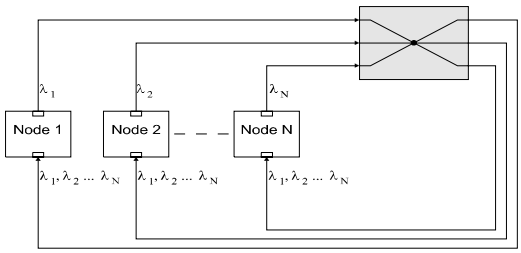

Figure 11: WDM network cluster.

5.2 Fiber-ribbon ring network

Although the WDM star architecture is very attractive and scales well to hundreds of these high-performance processing nodes, systems which require only a few tens of nodes can be realized by using optical fiber-ribbon links. Fiber-ribbon links offering an aggregated bandwidth of 8 Gb/s have already reached the market [14], and the price performance ratio is very promising.

Motorola OPTOBUS^{TM} bidirectional links with ten fibers per direction are used, but the links are arranged in a uni-directional ring architecture (Figure 12) where only *M*/2 bidirectional links are needed to close a ring of *M* nodes. The first nine fibers in each link form a bit-parallel high-speed channel, eight fibers for data and one fiber for clocking. All high-speed channels, together, form a high-speed ring network for circuit switched traffic. The access is divided into time-slots. However, in each slot the network can be divided into segments to allow for concurrent transmissions. High throughputs can therefore be achieved, especially in systems where some kind of pipelined data flow between the nodes dominates, like in a radar signal processing system.

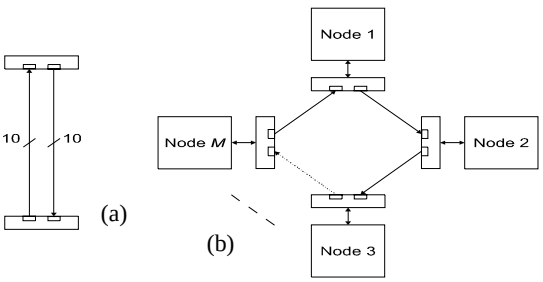

Figure 12: (a) Bi-directional fiber-ribbon link. (b) Unidirectional ring network.

The tenth fibers in all links together form a ring network dedicated for packet-switched traffic. When using, e.g, a token ring protocol on the packet network, this network will support low latency communication for sporadic packets at moderate traffic rates. At the same time it is assured that the circuit-switched traffic (typically the data flow in the signal processing chain) is not disturbed by packet-switched traffic.

Using links with 800 Mb/s per fiber translates to a bandwidth of 6.4 Gb/s for circuit-switched traffic (on eight fibers) and a bandwidth of 800 Mb/s for packet-switched traffic (on one fiber). This is enough for many systems, but for high-end systems, links with higher bandwidths – which are expected – are needed.

6. Conclusions

We have shown that the VEGA architecture is feasible for sensor array signal processing, it has the flexibility and scalability required in multi-purpose systems, the communication network has enough throughput and low latency to connect the processing modules, the mesh organization of the processor modules is suitable in terms of generality and scalability, and there can be a gain in efficiency and performance in building small PE arrays with high performance PEs instead of large arrays of low performance PEs.

7. References

[1] A. Åhlander, M. Taveniku, and B. Svensson, “A multiple SIMD approach to radar signal processing”, *Proc. IEEE Region Ten Conf., TENCON’96*, Perth, Australia, Nov., 1996, pp. 852-57.

[2] M. Taveniku, A. Åhlander, M. Jonsson, and B. Svensson, “A multiple SIMD mesh architecture for multi-channel radar processing”, *Proc. Intl. Conf. on Signal Processing Applications & Technology, ICSPAT’96*, Boston, MA, Oct., 1996, pp. 1421-27.

[3] J.G. McWhirter, “Algorithmic engineering in adaptive signal processing”, *IEE Proceedings-F*, vol. 139, no. 3, June 1992.

[4] C.M. Rader, “VLSI systolic arrays for adaptive nulling”, *IEEE Signal Processing Magazine*, July, 1996.

[5] Mercury Computer Systems Inc., http://www.mc.com/technology.html, World Wide Web.

[6] L. Bengtsson, “REMAP-γ: A scalable SIMD VLSI-architecture with hierarchical control”, Ph.D. Thesis, Chalmers University of Technology, 1997.

[7] R. Björklund and J. Olsson, “Design and implementation of processing elements in massively parallel processor arrays”, Masters Thesis, Department of Computer Engineering, Chalmers University of Technology, Sweden, 1996.

[8] M. Taveniku and A. Åhlander, “Instruction statistics in array signal processing”, Tech. Rep., Centre for Computer Systems Architecture, Halmstad University, Sweden, 1997.

[9] N. Quach and M.J. Flynn, “Suggestions for implementing a fast IEEE multiply-add-fused instruction”, Stanford University Technical Report CSL-TR-91-483.

[10] J.G. McWhirter, “Algorithmic engineering in adaptive signal processing”, *IEE Proc.*, Pt. F, Vol.139, No.3, June 1992.

[11] S.F. Oberman, M.J. Flynn, “An analysis of division algorithms and implementations”, Tech. Rep. CSL-TR-95-675, Stanford University, 1995.

[12] P. Soderquist and M. Leeser, “An area/performance comparison of subtractive and multiplicative divide/square root implementations”, *Proc. 12th Symp. on Computer Arithmetic*, pp. 132-, 1995.

[13] M. Jonsson, A. Åhlander, M. Taveniku, and B. Svensson, "Time-deterministic WDM star network for massively parallel computing in radar systems", *Proc. Massively Parallel Processing using Optical Interconnections, MPPOI’96*, Lahaina, HI, USA, Oct. 27-29, 1996, pp. 85-93.

[14] OPTOBUS Home Page, http://design-net.com/logic/optobus/homepage.html, World Wide Web.

[15] M. Jonsson, B. Svensson, M. Taveniku, and A. Åhlander, "Fiber-ribbon pipeline ring network for high performance distributed computing systems", *Proc. Int. Symp. on Parallel Architectures, Algorithms and Networks, I-SPAN’97*, Taipei, Taiwan, Dec. 18-20, 1997.