A Scalable VLSI MIMD Routing Cell

H. Corporaal J.G.E. Olk

Delft University of Technology
Faculty of Electrical Engineering
Section Computer Architecture and Digital Systems
Mekelweg 4, P.O. Box 5031
2600 GA Delft, The Netherlands
email: heco@duteca.et.tudelft.nl eddy@duteca.et.tudelft.nl

Abstract

It is a well known fact that full custom designed computer architectures can achieve much higher performance for specific applications than general purpose computers. This performance has to be paid for: a long design trajectory results in a high cost-performance ratio. Current VLSI design and compilation tools however, make semi-custom designs feasible with greatly reduced costs and time to market.

This paper presents a scalable and flexible communication processor for message passing MIMD systems. This communication processor is implemented as a parametrisized VLSI routing cell in a VLSI compilation system. This cell fits into the SCARCE RISC processor framework [1], which is an architectural framework for automatic generation of application specific processors.

By use of application analysis, the cell is tuned to the specific requirements during silicon compilation time. This approach is new, in that it avoids the general performance penalty paid for required flexibility.

Keywords *communication processor, fine-grain communication, virtual connections, routing, scalability of design, message compression*

1 Introduction

Massive parallel MIMD processing is a promising approach to satisfy the needs for computing power. Especially the peak performance of message passing computers is almost unlimitedly extensible by adding more computing nodes to the network of processors.

Not every application is suitable for (massive) parallel computing, because of inherent serial algorithmic components which dominate the application. The success of sequential supercomputers is partly related to that. Fortunately, large classes of applications, and among them those which require processing power most, can easily be mapped on a regular network of processing nodes. However, the communication requirements of these applications can be quite different. These requirements include aspects like the communication-computation bandwidth ratio, the node degree (nr. of links per node), requirement of fixed connections, the average message size, burst communication etc.

Recently many communication processor proposals and designs are made, supporting message passing in massive parallel systems [3,4,5,6,7,8,9,10,11]. In [2] we explored the design space for building such processors. It turned out that many designs cover a very restricted area into this design space, and are therefore only suitable for very limited application areas.

This paper presents a scalable and flexible communication processor for message passing MIMD systems. This communication processor is implemented as a parametrisized VLSI routing cell in a VLSI compilation system[1]. This cell fits into the SCARCE RISC processor framework [1], which is an architectural framework for automatic generation of application specific processors. The communication processor offers MIMD capabilities to this framework, which broadens the application domain.

Using application parameters we can quickly generate an application specific MIMD processor, avoiding the long design trajectory inherent to full custom design, without sacrificing performance too much.

Because the communication processor (CP) is integrated with the data processor (DP) we do not have to pay for the usual overhead required to start a send or receive operation. By using a register mapped interface to the DP fine grain communication will be supported very

[1] We use the ASA VLSI compiler system of SAGANTEC

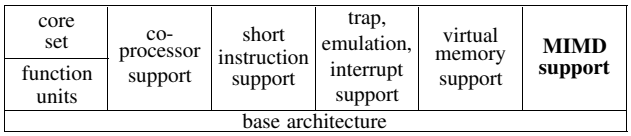


Figure 1: SCARCE architecture framework with additional MIMD support.

efficiently. On top of that we implemented a mixture of wormhole routing and virtual links. This allows for low latency communication, high link utilization, multiple virtual connections and adaptive intermediate node routing.

Together, this makes this processor well suited for systolic like applications. These applications are characterized by a regular data flow between (near) neighbor nodes, requiring very high communication bandwidth on dedicated channels. The communication bandwidth has to be well balanced with the processing bandwidth. Because of the scalability and flexibility, the described design is also suitable for more general classes of computations.

This paper first briefly describes the SCARCE framework and the place of the routing cell within this framework. Then possible scalability and flexibility aspects of communication processors are discussed. Next we describe the scalable communication processor framework, which is currently under design. The design choices made with respect to the design space described in [2] are clarified and the implemented scalability and flexibility is listed. Then an overview is given of the functional blocks of the CP. A status overview section is directed to the currently implemented CP. It also points at the research done to complete the application analysis and automatic processor synthesis framework.

2 SCARCE framework

The SCARCE framework [1] allows automatic generation of application specific VLIW (Very Long Instruction Word) processors and embedded controllers. The addition of a parametrizable VLSI routing cell to SCARCE gives the framework MIMD support and allows processor generation for a wider range of applications, especially for the fields of signal and image processing. The base architecture components, including the added MIMD support, of the SCARCE framework are shown in Figure 1.

This proposal of a parametrized basic VLSI routing cell for SCARCE defines a space of possible CP implementations. The resulting CP architecture is specified at chip design-time using application analysis. The application characteristics are translated to cell parameters which are used to generate (from a generic description) a hardware description in SID[2] of the routing cell. The actual chip layout is obtained by silicon compilation with the ASA silicon compiler. The resulting routing cell resides on the same chip as the SCARCE data processor, which is also similarly defined at chip design-time. The platform for generating application targeted SCARCE MIMD architectures is pictured in Figure 2.

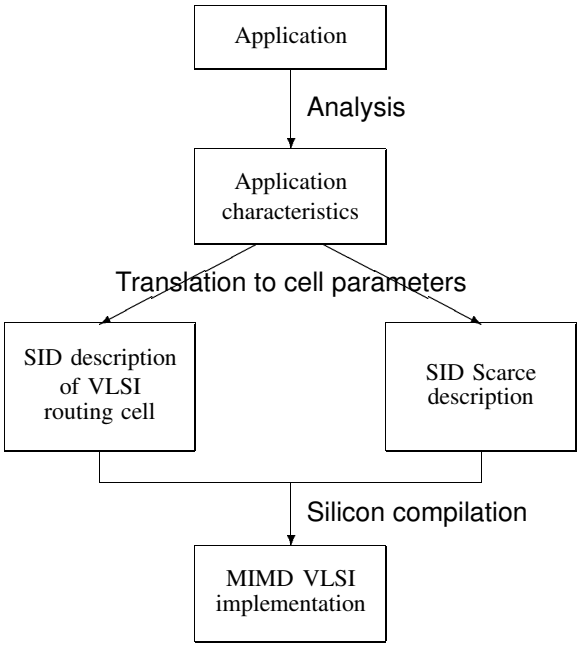


Figure 2: Platform for generation of application specific SCARCE MIMD architectures.

3 Scalability and flexibility

This section of the paper defines scalability and flexibility and considers the various ways of implementing scalability and flexibility.

Scalability and flexibility are essential to permit the adaptation of the CP design to specific applications, which is a more cost effective way than designing a completely new CP architecture for each application. Scalability is applied at chip design-time or board design-time. At compile time and run time, flexibility of CP design characteristics is possible. Since scalability is part of the design

[2] Sagantec Input Description is a hardware description language, comparable to VHDL, used in combination with the ASA silicon compiler platform.

trajectory it offers more radical variations of design parameters than flexibility, which only allows alteration of parameters after the design-time.

When considering the design model for communication processors (see [2]), a distinction can be made between two ways of scalability:

1. **Scalability of the network topology:** by alteration of the degree of the CP (i.e. the number of links), the number of nodes in the network or the number of virtual networks.

2. **Scalability of performance:** by adjustment of link bandwidth, internal CP datapath, buffer size and implementation technology.

Flexibility at compile and run time of design characteristics is may be supported by:

- **Virtual topology:** although the physical topology of the network can not be altered, the virtual topology can be changed however. For instance, different virtual topologies may be mapped onto the same physical topology.

- **Routing:** the routing path from source to destination taken by a message may be chosen differently. Or arbitrary topologies can be routed using e.g. lookup tables, interval routing, streetsign routing (see [2]).

- **Switching method:** wormhole switching, described in [3], allows emulation of packet and circuit switching methods.

- **Message control:** the exchange of messages between the DP and the CP can be accomplished in several ways. A CP may use DMA to deliver or send a message, but also the DP can be employed for message transfer.

4 Design of the routing cell

This section discusses the choices made for the routing cell design parameters, with respect to the design model of [2], and clarifies the scalability and flexibility issues of the previous section. The scalability and flexibility offered by the parametrized VLSI routing cell is summarized in Table 1 and Table 2 respectively. Subsequently the following design parameters are reviewed: (virtual) network, routing, deadlock & livelock, switching method, message structure, buffering, interfacing and fault tolerance.

parameter	range/type
degree	arbitrary[a], ≥ 1
no. nodes in network	arbitrary[b], $1 - 2^{FD}$
no. of virtual networks	arbitrary, ≥ 1
link implementation	optional priority mechanism (a)synchronous, uni-,bidirectional, decoded/encoded control
internal datapath	1-n bits
external (link) datapath	1-n bits
data compression	optional
buffer size	arbitrary[c]

[a] At chip design-time, or at board design-time.
[b] F = wordsize field, D = degree.
[c] Router enforces lower bound on size of input buffer.

Table 1: Design scalability.

parameter	range/type
virtual topology	alterable
routing	extended E-cube, intermediate node routing
switching method	wormhole ($\ni$ circuit + packet)

Table 2: Routing cell flexibility.

4.1 Network

Central to our CP design is the use of coupled virtual networks (VNs) sharing the same physical links, pictured in Figure 3. Virtual networks have several advantages:

- Improved link utilization since blocked messages don't consume physical link bandwidth and other virtual links can use this unused bandwidth of the idle virtual links.

- Allows irregular routing when using a deterministic routing algorithm (e.g. E-cube). Irregular routing can be implemented as intermediate node routing. Examples of possible uses are: routing in manhattan

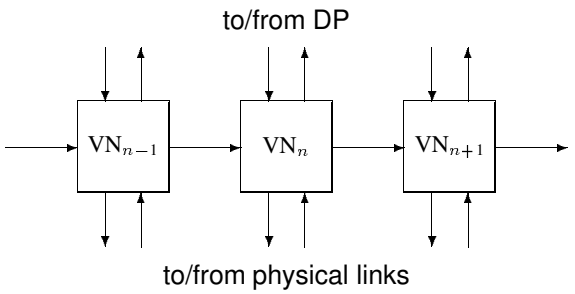
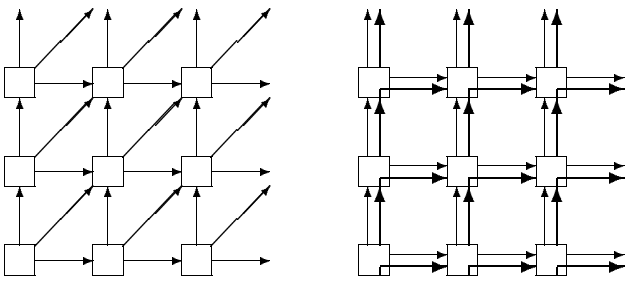


Figure 3: Coupled virtual networks.



Typical systolic communication pattern

Systolic communication using virtual links

Figure 4: Use of virtual links for a typical systolic communication pattern.

networks, routing to irregular connected I/O-nodes, random node routing (to avoid hot spots).

- Implementation of high priority virtual networks for system messages, which allows easy and fast exception handling (no need for saving/restoring whole CPU state).
- Virtual networks are easy implemented using simple (modular) hardware which makes use of high clock frequencies possible and thus achieving high performance. Additionally, pipe-lined schemes may be employed in the future to reduce cycle-time and increase throughput.
- Virtual topologies can be mapped on a different physical topology, e.g. for systolic communication (see Figure 4). Also static fault tolerance (e.g. routing around faulty nodes) is simplified using virtual topology mapping.

The routing cell supports the construction of multi-dimensional mesh networks suiting most applications, particularly when using virtual topology mapping. The degree is arbitrary and the routing cell features an arbitrary number of virtual networks (limited by chipsize only). Also multiple CPs may be chained to increase the degree of a node and the number of virtual networks.

4.2 Routing

The CP uses extended E-cube routing, a deterministic deadlock-free routing scheme extended to virtual networks. That means that both the links within a virtual network, as well as the virtual networks themselves, are traversed in fixed order. This scheme combines simplicity and adaptivity. Extended E-cube routing is very easy and efficient to implement. On the other hand we still have the possibility of source node controlled adaptive routing.

4.3 Deadlock & livelock

Deadlock within a virtual network is prevented by using extended E-cube routing. Deadlock between virtual networks is avoided by letting messages traverse the virtual networks in fixed order. Livelock is circumvented by fairly sharing the resources (buffer space, link bandwidth, crossbar connections), so no message can wait forever to gain access to a resource. This is sufficient in our case where messages are always (logically) routed towards their destination.

4.4 Switching method

A wormhole switching method is used allowing emulation of both circuit (using long messages) and packet switching (using short messages). Virtual links (and their virtual networks) are used to support systolic communication, establishing multiple connections between different nodes.

4.5 Message structure

A message consists of a varisized address part and a varisized data part. This offers the most flexibility without severe performance penalties. Both parts consist of *fields*, the data unit transported over the networks. The wordsize of a *field* is parametrizable. The first implementation uses a 32-bit field, which suits the 32-bit RISC architecture of SCARCE.

The address part of a message consists of one or more relative address *field(s)*, one *field* for each dimension of the network. These dimensions are considered unidirectional, so a grid with (logical) bidirectional links has four dimensions: left, right, up, down. Each dimension *field* holds a positive value and is chopped off when it becomes zero, i.e. the message (the address part at least) has finished traversing over the corresponding dimension. The varisized address part is followed by a likewise varisized data part which exists of one or more data *field(s)*. Both the last data and address fields are indicated by a so called end-of-type bit.

A *field* is divided in several so called *flits* [5]. The number of *flits* in a *field* depends on the width of the internal datapath and whether the simple data compression scheme (based on sign extension) is used. The last *flit* of a *field* is specified using an end-of-field bit. The data compression reduces the consumed bandwidth of, among other things, the physical links. Here we use the property that a large percentage of data consists of small values. This is especially true for relative addresses in near-neighbor communication. Using e.g. 4-bit *flits*, a *field* may contain up to 8 *flits*.

4.6 Buffering

Distributed buffering is used, i.e. separate buffers for each link. Although distributed buffering, compared to central buffering, uses the total available buffer space inefficiently, it not only makes deadlock prevention straightforward but also buffer management easier and faster.

4.7 Interfacing

The interfacing of the CP to its outside world falls apart in interfacing to the on-chip SCARCE DP and interfacing to the neighboring CPs.

DP-CP interface: The data transferring between the DP and CP is register interfaced and takes place, implicitly, by reading/writing from/to registers. From the DPs point of view these registers are not different from the general purpose registers but have the side effect of sending or receipt of data to/from the CP. One combined send/receive register is necessary for each virtual network. Additionally, a status/command register is needed to determine whether data can be read/written from/to the virtual networks and to specify what is actually send (address or data fields).

Using a register interface for transferring data between CP and DP is ideal for systolic applications since it provides fast communication and allows to perform arithmetic operations directly on incoming data. For instance, just one instruction (e.g. ADD R_{r1},$R_{r2} \rightarrow R_{s1}$) is needed to receive two 32-bit words, perform an add operation and send the result back over the network.

However, register interfacing requires certain characteristics of the architecture framework. The framework must support locking and exceptions. Locking must occur when an empty receive register is read or a full send register is written. With locking the current instruction is simply started again until the lock is inactivated. Since the maximum time a lock situation will exist is unknown, it is desired to generate an exception after the lock situation has been active for a certain amount of time. The DP can then execute other processes and will try to execute the lock causing instruction later. Especially with pipelined architectures (such as SCARCE) locking and exception mechanisms are far from trivial.

CP-CP interface: The physical link interface between neighboring nodes is generally a trade-off between pin usage and desired performance. This highly depends on the targeted application and therefore the VLSI cell supports the use of various physical link implementations: using either unidirectional or bidirectional pins and an asynchronous or synchronous data transfer protocol. Also multiplexing of (flow) control and data signals is supported to reduce pin usage with limited performance loss.

In combination with this encoded control, a loose flow control scheme is used which means only (receiver buffer) status changes are transmitted, i.e. when an input buffer threatens to become full and when, after the buffer was full, enough buffer space (e.g. half the buffer size) is available again. So when buffer usage is balanced, no control flow is necessary and encoded control may provide a more efficient use of pins for data transfers than decoded control. Of course, with decoded control receiver buffer status is constantly available to the sender at the expense of separate pins.

Table 3 lists, for different link implementations, (1) the number of pins per link for N virtual networks and a data transfer rate of M data bits per cycle and (2) the effective pin usage (used for data transfers) for various traffic distributions: (a) unidirectional, all traffic in one direction, (b) 75% of traffic in one direction, 25% in opposite direction and (c) balanced bidirectional traffic (50% traffic in either direction). The shown values for links with encoded control vary since they depend on the flow control overhead, both worst cases (maximum overhead) and best cases (no overhead) are listed. From the performance figures it emerges that there exists no a priori favorite physical link implementation.

4.8 Fault tolerance

The first implementation doesn't feature any provision for dynamic fault tolerance. However, static fault tolerance is possible. Additional virtual networks may be used to route around faulty nodes or the desired virtual topology may be remapped on the (now changed) physical network.

5 Functional overview routing cell

This section briefly discusses the various functional components of the VLSI routing cell. A block diagram of the cell is given in Figure 5 showing two virtual networks and two link interfaces (thus degree is 2). In order to maintain a clear picture, the interfacing to the DP is omitted from the block diagram.

The two connected networks have various input and output channels. Data can be received over input channels from the DP interface, a physical input link, or from a previous network (if present). Data can be sent via output channels to the DP interface, a physical output link, or to the next network (if present). Each network consists of: one router (incorporating an input buffer) for each incoming channel, one (optional) output buffer for each

implementation			number of pins	effective pin usage for traffic pattern:[a]		
pin type	control	transfer	per link[b]	A	B	C
unidir	decoded	sync	$M + 2 + \log N + N$	50.0%	22.2%	50.0%
unidir	decoded	async	$M + 4 + \log N + N$	44.4%	19.8%	44.4%
unidir	encoded	sync	$M + 3$	18.2-36.4%	18.2-32.3%	18.2-72.7%
unidir	encoded	async	$M + 5$	15.4-30.8%	15.4-27.4%	15.4-61.5%
bidir	decoded	sync	$M + 3 + \log N + N$	47.1%	47.1%	47.1%
bidir	decoded	async	$M + 5 + \log N + N$	42.1%	42.1%	42.1%
bidir	encoded	sync	$M + 4$	16.7-66.7%	16.7-66.7%	16.7-66.7%
bidir	encoded	async	$M + 6$	14.3-57.1%	14.3-57.1%	14.3-57.1%

[a] For typical case with: 4 virtual networks (N) and datawidth of 8 bits (M). A=traffic in single direction, B=25% of total traffic in one direction, C=50% of traffic in either direction.
[b] M=number of data bits transferred per clock cycle, N=number of virtual networks.

Table 3: Performance of link implementations.

outgoing channel and a crossbar to physically connect input channels with output channels.

The input buffers store (part of) the fields coming in over the virtual input links. They also provide the associated routers with information to determine the output links the messages must be routed to. Since an established crossbar switch can be viewed as a plain wire, output buffers are not strictly necessary (and thus optional) to buffer waiting flits for a virtual output link. However, they have the slight advantage that the crossbar switch can possibly be reconfigured sooner for a following message. Since the last flits of a message can be stored in the output buffer while waiting to be transferred over the link, the crossbar switch can already be switched to another output link for the trailing (head of a) message. This is most advantageous for short messages, i.e. packet switching alike communication.

Input buffers may have arbitrary buffer space, although there is a lower bound since the router must be able to examine one or more address flits. Output buffers may have arbitrary buffer space. Furthermore, when a buffer is empty, it is transparent and can be regarded as a plain wire; it has some propagation delay but no additional clock cycle delay.

The router examines incoming address flits, requests the appropriate output channel and waits until connection over the crossbar is established. Then, as the flits starts to move across the crossbar, the router decrements the first address field and waits until the last data field passes. It then relinquishes the connection and waits for the first address field of the next message to arrive.

Since the router needs information on more than one flit to determine the outgoing channel, it must be able to examine multiple flits in the input buffer, which is therefore incorporated in the router. When data compression of fields is used, the router only has to examine a single flit since the zero value is represented by one flit and the router just needs to know whether the address field is zero or not.

A crossbar receives requests for output channels from the routers and services these requests using a round-robin scheme. Because of (extended) E-cube routing, the crossbar need not be fully connected which saves some chip area.

The shown link interfaces provide the two networks with virtual connections to neighboring nodes. A link interface consists of an input link and an output link. The input link forwards data coming in over the physical link to the corresponding network while an output link time-multiplexes data from the networks over the physical link using a round-robin scheme. However, some networks may have higher priority than others, e.g. for system messages.

For most physical link implementations, one input link and one output link together form a (logically) bidirectional link. Only a physical link with unidirectional pins and decoded control signals allows independent use of input and output links. In all other cases control information, regarding the flow control, must be exchanged between the input link and the output link of a physical link.

6 Status and current research

This section gives an overview of the implementation status of the VLSI cell and the related research. At the moment a functional and structural implementation in SID of the first version of the generic routing cell has just been completed. The characteristics for one specific realization

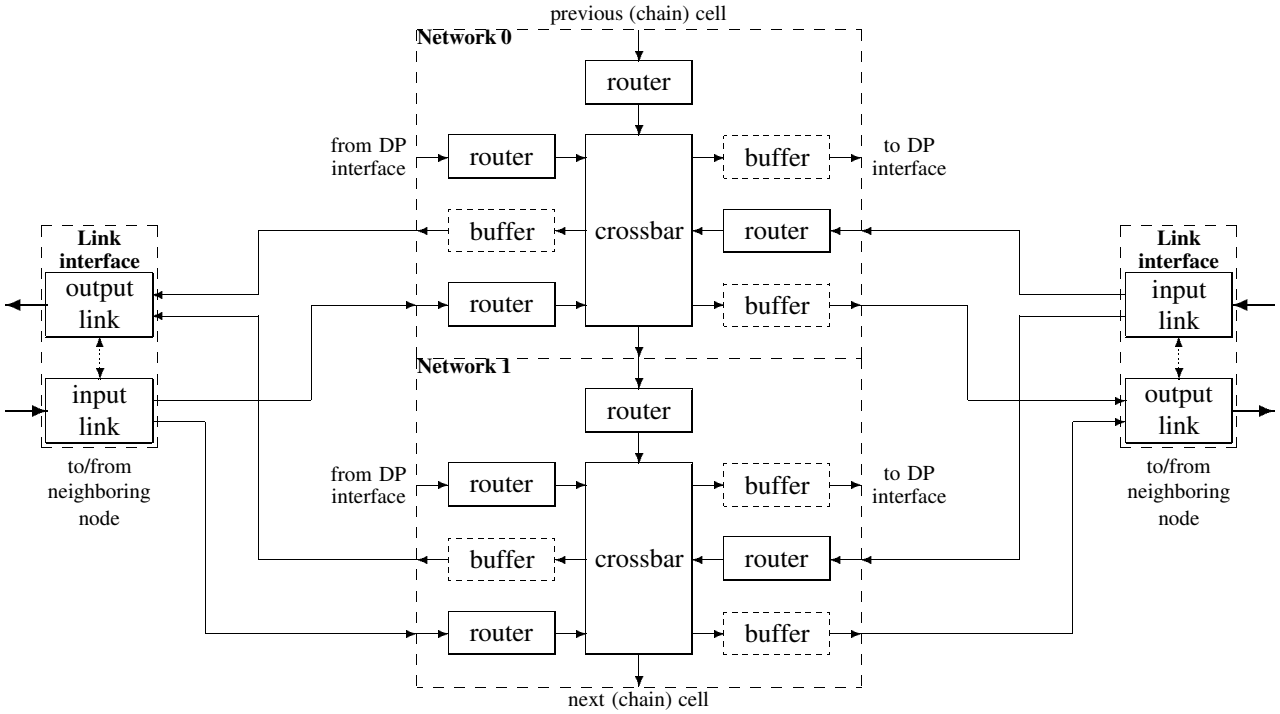


Figure 5: Block diagram of routing cell.

of the VLSI routing cell featuring four virtual networks, a degree of four and unidirectional physical links with decoded control signals, are summarized in Table 4.

Besides implementation of the VLSI routing cell, research is being done focussing on the project goal, already pictured in Figure 2, which provides an addition to the SCARCE application analysis tools: automatic synthesis of an application to get the most optimal routing cell for the specific application. The research includes:

- Implementation and addition of the various physical link realizations to the parametrized VLSI cell.
- Adding broadcast capabilities and absolute addressing to the VLSI cell.
- Design of regular structures (partly full custom), e.g. for buffers, to be used by the silicon compiler in order to decrease the occupied chip area.
- Application analysis: analysis of application characteristics and translation of these characteristics into values for design parameters of the VLSI cell. Besides the parameters of the VLSI cell, other characteristics are needed, for instance of the topology.
- Determination of efficiency of random node routing in avoiding hot spots for more general MIMD applications.
- Determining effectiveness of data compression using simulation of real systolic applications.

This research requires an integrated MIMD simulation and analysis tool, which incorporates the SCARCE architecture framework and the CP parametrisized architecture. The GNU gcc/c++ compiler platform, extended with asynchronous communication primitives, will be used for code generation.

7 Conclusions

This paper discussed the design of a parametrisized VLSI cell for MIMD routing support, which can be tailored to application specific needs. This design is integrated into the SCARCE architecture framework for the automatic synthesis of application specific processors. We investigated the scalability and flexibility issues, in order to maximize the design space covered by our CP implementation, without sacrificing speed. By translating of the application characteristics into design parameters, the integrated data and communication processor VLSI cell can be automatically generated by our silicon compiler system. Although a time saving silicon compiler is used, part(s) of the chip still has/have to be designed full custom to attain a reasonably compact layout.

Design aspects		Performance aspects	
parameter	value(s)	parameter	value
1. Network	degree: 4, $N \times N$ mesh, 4 virtual networks	Total number of link pins	128
2. Routing path	partly nondeterministic, controlled by CP/DP	Total throughput	8 bytes/cycle
3. Deadlock & livelock	prevention	Throughput per link	1 byte/cycle
4. Switching method	wormhole	Compression ratio	1-4
5. Message structure	relative address, varisized address & data fields data compression	Hop latency	$\geq$1 clock cycle
		Router decision	$\geq$2 clock cycles
6. Buffering	distributed buffers	Interface registers	4 send, 4 receive, 1 status/command
7. CP-CP interface	8 bits unidirectional, decoded control signals		
8. DP-CP interface	register mapped		

Table 4: First routing cell implementation characteristics.

This routing cell supports multidimensional arrays and has a simple, but flexible, routing strategy using intermediate node routing, using two or more of the available virtual networks.

Scalability in performance is achieved by freedom of physical link implementation, internal datapath width, internal buffer size and implementation technology. We can further change the number of virtual links, and choose between uni- and bidirectional links. A very simple and fast data compression scheme is used to reduce bandwidth usage of small data and address values.

The scalable design of the SCARCE VLSI routing cell supports, but is not limited to, systolic fine grain communication, with very low communication overhead and combines the speed of fully dedicated systolic hardware with the flexibility and quick prototyping of general purpose computing. It allows simple design of application specific massive parallel processor systems and therefore significantly shortens the design process of such systems.

References

[1] J.M. Mulder et al. A framework for application-specific architecture design. In *Proceedings of the 14th International Symphosium on Computer Architecture*, May 1989.

[2] H. Corporaal and J.G.E. Olk. Design and evaluation of communication processors supporting message passing in distributed memory systems. In *Proceedings of the 6th Distributed Memory Computing Conference*, April 1991.

[3] Charles L. Seitz et al. *The Hypercube Communications Chip*. Technical Report, Dept. of Computer Science, California Institute of Technology, 1985.

[4] Dick Pountain. Virtual channels: the next generation of transputers. *BYTE Magazine (International Edition)*, 15(4):E&W 3–12, April 1990.

[5] William J. Dally. *A VLSI Architecture for concurrent Data Structures*. Kluwer Academic Publishers, 1987.

[6] J. K. Annot and R. A. H. van Twist. A novel deadlock free and starvation free packet switching communication processor. In *Parallel Architectures and Languages Europe*, page 68, 1987.

[7] Daniel A. Reed and Dirk C. Gronwald. The performance of multicomputer interconnection networks. *IEEE computer*, June 1987.

[8] Daniel Hillis. *The Connection Machine*. MIT Press, 1985.

[9] e.a. Borkar S. Iwarp: an integrated solution to high-speed parallel computing. In *Proceedings of Supercomputing '88*, 1988.

[10] W.G.P. Mooij. *Packet Switched Communication Networks for Multi-Processor Systems*. PhD thesis, University of Amsterdam, 1989.

[11] C. Germain et al. A new communication design for massively parallel message passing computers. In *IFIP Working Congerence on Decentralized Systems*, Lyon, December 1989.