Application Driven MIMD Communication Processor Design

Eddy Olk and Henk Corporaal

Department of Electrical Engineering, Delft University of Technology,
P.O.Box 5031, 2600 GA Delft, The Netherlands

The design space of communication processors for message passing distributed memory systems is rather large. Many of the design decisions that are taken seem to be dependent on the targeted application, or application area, considering the various different communication hardware designs, commercial as well as research based, that have been presented. This paper describes a framework currently under development that aims, given a specific application, at obtaining cost effective communication hardware for message passing distributed memory systems.

Keywords: MIMD simulation, scalable communication processor, message passing.

1. Introduction

Massively parallel computers consist of many (100s – 1000s) processing nodes that are interconnected via a communication network in order to solve a specific problem cooperately. Most promising parallel computer architectures to exploit available parallelism in applications are message passing distributed memory (MIMD, Multiple Data Multiple Instruction) systems [1]. The processing nodes, each node having its own local memory, communicate by sending messages over the network. As pictured in Figure 1 special communication hardware is present; besides the data processor DP and the local memory MEM, each node also includes a communication processor CP. The function of the CP is to provide a fast transparent[1] communication mechanism to allow each node to communicate with every other node in the network. Yet, the design of these communication processors is not straightforward and many design decisions need to be made [2]. Most of these decisions are a tradeoff between offered performance and

[1]The large number of nodes in the network makes it practically infeasible to connect every node in the network directly with every other node; communication thus may take place via intermediate nodes.

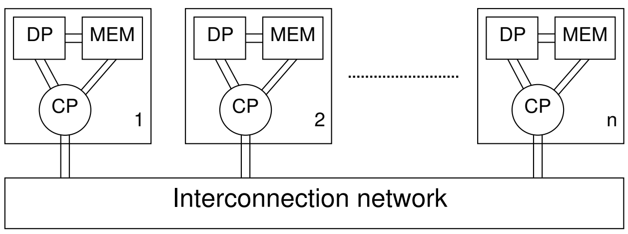

Figure 1. Each node in the network consists of a data processor DP, local memory MEM, and a communication processor CP.

hardware implementation cost and complexity. This is strongly dependent on targeted application domains as is illustrated by the many different communication processor designs for message passing MIMD systems in commercial designs, as well as in the research field.

In order to design cost effective communication hardware suiting a specific application (domain), a simulator framework named PARSE (Parallel ARchitecture Simulation Environment) is being developed for analysis of parallel MIMD applications. The PARSE framework is an extension to the MOVE framework [3] that aims at generation of application specific processors with a RISC, VLIW or transport triggered architecture. The remainder of the paper will discuss design and implementation aspects of the PARSE framework.

Section 2 gives an overview of the PARSE framework. The communication hardware architecture modeled by the framework is described in Section 3. Performance versus hardware implementation cost aspects of communication processors are discussed in Section 4. The implementation of the architecture simulator is briefly discussed in Section 5. Some results obtained with the framework are presented and discussed in Section 6. Finally, Section 7 concludes the paper.

2. PARSE framework overview

Figure 2 shows the top level view of the PARSE framework for analysing the communication of parallel message passing MIMD applications. The goal of the framework is to obtain design parameters that characterise the most cost effective communication hardware required to meet the targeted application's performance requirements. This requires that the application is actually executed since its communication behaviour may be data dependent.

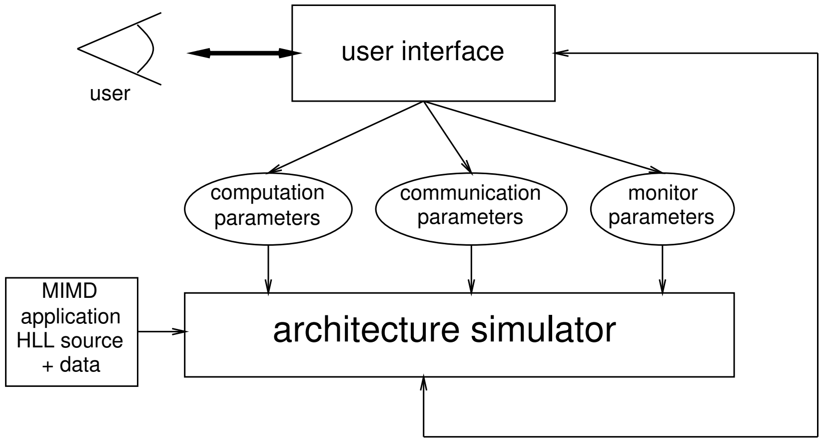

Figure 2. Overview of the PARSE framework.

The framework consists of a parallel architecture simulator that simulates a MIMD architecture and can run a parallel application, specified in a High-Level Language (HLL), with its own application data. The simulator accurately models the communication hardware to obtain realistic performance results. Analytic communication network performance models do generally not, or insufficiently, account for certain aspects which may effect overall performance, e.g. hot spots in the network where parts of the network are heavily congested. Advantages of using a simulator to analyse the application instead of running it on a real parallel system are:

- No actual real parallel system is required, simulation can be run on a sequential system.
- Measuring on a real system is difficult since most aspects are inaccessible for measurement.
- No *probe* effect when simulating an application. Real execution behaviour may be influenced by added measurement code.
- Easy to change architecture parameters, e.g. the network topology.
- Consistency (repeatability) between simulator runs. Interesting phenomena can be examined more closely by rerunning the simulation.

Obviously, simulation is slower than real execution. However, the object oriented implementation of the simulator [4] makes development of a parallel executing simulator relatively easy. Running the simulator on a general purpose parallel computer will be advantageous over a hardware prototyping approach since it requires no infrastructure for hardware development and allows easier modification of architecture parameters.

The user communicates with the simulator using a user interface which shows simulator and application output, allows changing the architecture parameters and lets the user indicate the aspects of the parallel system architecture to be monitored, i.e. the measure points (e.g. buffer utilisation, link utilisation, message latency). Two categories of architecture parameters can be changed:

1. **Computation parameters:** defining the performance of each node in the system (e.g. data processor cycle time)
2. **Communication parameters:** defining the architecture and performance of the communication network (e.g. routing algorithm, network topology, sizes of buffers, number of virtual links, routing latency, link bandwidth).

The user can iterate towards an 'optimal' architecture by analysing the simulation results of various application runs with varying parameter values and application data sets. Analysis of the simulation results may also point at inefficiencies of the application's implementation, e.g. a bad (with respect to communication behaviour) mapping of processes on processors. The user may then decide to revise the implementation of the application.

The simulation results give the user the total execution time of the whole simulation as an indication of how well the application runs on the configured architecture. In order to help the user in the design process, some other information is provided as well:

- **Communication:** (total) amount of communicated data. A list is given with the most communicating (pair of) nodes.
- **Link bandwidth usage:** percentage of the available bandwidth of links between neighbouring nodes actually used for data transfer. The average (global) bandwidth usage is given together with a list of links having the highest bandwidth consumption.
- **Buffer usage:** maximum percentage of available buffer space at each node actually consumed. The nodes with the highest maximum buffer usage are listed.
- **Hot spots:** nodes are listed which have a high occurrence of data congestion, i.e. where data from different directions concurrently wants to traverse over the same link.

In addition to the information listed above, the architecture simulator can generate a trace file containing traces of simulation events (user specified) that can be used for detailed analysis without running the application again.

3. Communication architecture

The framework covers the most interesting part of the design space of communication processors by using a communication architecture that has design parameters, according to the classification model of [2], as summarised in Table 1. This section does not discuss the parameters in detail, it just gives a brief description of the communication architecture.

Table 1
Important layer design parameters of CP architecture implemented by the framework.

layer	**parameter**	**values**
transport	entity representation	
	addressing mode	absolute
	address field, data field	variable size
network	entity representation	
	addressing mode	absolute
	address field, data field	variable size
	buffering	centralised
	routing	
	path control	network
	path	deterministic
	collision resolvement	buffered blocking
	network support	
	degree	$1 - n$
	topology	multidimensional meshes, k-ary n-cubes
	deadlock resolvement	acyclic
	livelock resolvement	fair sharing
data link	virtual layer support	yes
	entity representation	
	data field	8, 16, 32, 64 bits
	flow control	
	granularity	tight
	window	≥ 1

The data flow diagram of the architecture of the communication processor is pictured in Figure 3. The CP has a single *register file*, to temporarily store data coming from neighbouring CPs, or the DP, over the *input links*. (It serves as buffer space.) Within the register file, each input link has (and manages) its own register space to store its incoming data. The register file is multiported so all input links can write to it simultaneously and also all *output links* can read from the register file concurrently in order to send data to the neighbouring CPs. The FIFO (First In First Out) buffers positioned between the DP and the CP improve performance by allowing the DP to read/write small amounts of data fast with minimal chance on blocking for every data element read/written.

Every physical link between CPs, as well as the DP link, is divided in *channels* or *virtual links*. Each channel can transport data independently and shares the available bandwidth with the other channels of the link. Virtual links help avoid deadlock problems and make more efficient use of the link possible.

In order to process the incoming communication data, i.e. messages, every input link has a *router* for each channel of the input link. Messages are composed of an address and a data part,

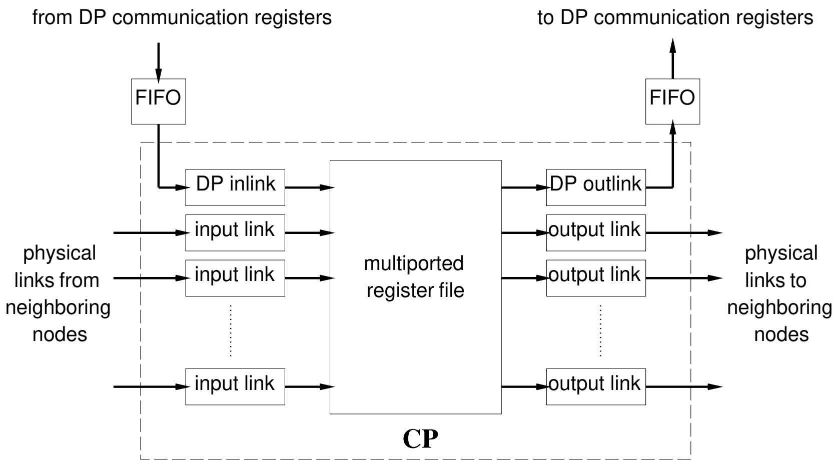

Figure 3. Data flow diagram of CP architecture used by simulator.

both having a variable number of *flits*[2]. The architecture uses wormhole routing [5] so (the flits of) a message may be spread out over various nodes. As soon as the address of an incoming message is known, the router examines the address and decides where this message must be routed to, i.e. to one of the output links or to the DP. The router then tries to allocate a free (unused) channel from the specific output link. When this succeeds, the router will keep the output link informed of the presence of message data for the specific channel. The output link will read the message data from the register file and send it over the link, thus forwarding the message.

4. Performance versus hardware cost

The performance of a parallel system is composed of the processing performance of each node and the performance of the communication network. Despite the fact that the PARSE framework is aimed at analysing communication performance, processing performance is important since it influences the communicational behaviour of the application. Yet, we do not want to model the architecture of the DP too accurately since this unnecessarily slows down the simulation. So we like to model the processing performance just accurate enough to yield the proper communicational behaviour. The architecture simulator assumes the processor architecture of each processing node is similar[3] to the actual processor executing the simulation [4]. The computation to communication performance ratio can then be altered by specifying the clock cycle time of the data processor.

The communication performance falls apart in two aspects: throughput and latency. Throughput, or bandwidth, is the amount of data that can be sent along a link or CP in a certain time. Latency is the time it takes data to move through a CP or traverse a link. Generally, the throughput of links and CPs are identical since it does not make sense to, for example, have high throughput links when each CP has a much lower bandwidth. So communication performance is characterised by the link throughput, the link latency (propagation delay) and the CP node

[2]A flit is the smallest data unit managed by flow control.

[3]For the performance model, the memory interface is assumed to be ideal so the effects of cache misses and page faults are ignored. Different architectures can be simulated by using different basic block costs (and control flow graph is unchanged).

latency (buffering, routing decision time). The framework has quite a number of parameters to allow the user to define the communication performance (see [4]).

If the communication hardware is integrated on a single chip, its implementation costs mainly depend on the chip die area and the number of chip package pins. Larger die area and more package pins result in higher costs. These costs exclude design costs and assume the data processor resides on a different chip. For a communication processor, the die area is mainly allocated to buffers, busses and crossbar switches [6]. So the total die area depends on: the number of buffers, the size of each buffer, the number of crossbar switches (related to the network topology), and the width of the data path. Chip pins are consumed by communication links and the interface to the DP. The total number of required pins thus depends on: number of links, number of pins per link, number of pins of the DP-CP interface.

There is, obviously, a relation between the amount of implemented communication hardware and the resulting performance. More hardware generally means increasing performance. For instance, the bandwidth of a link can be increased by using more pins. Examination of hardware resource usage (buffers, link bandwidth) learns whether increasing communication hardware and related performance is actually used completely.

5. Simulator implementation

This section briefly describes the implementation aspects of the parallel architecture simulator, a more extensive discussion of the simulator can be found in [4]. The simulator, implemented in C++, uses the *execution-driven* approach which means that it interleaves the execution of the application with the simulation of the multi-processor architecture to do performance analysis. So the simulator produces program results, as well as execution statistics from the simulated architecture.

The simulation is event-driven and translates all changes in the communication network to time-stamped events which are executed in time sequence. When no more events are to be executed, the simulation is finished. Furthermore, *direct execution* of application code is performed. Direct execution means that the application code's instructions are not simulated but directly executed and additional cycle-counting instructions are added (during compilation) to time the code. This is called *code augmentation*.

Applications that are to be run on the simulator must be written using the *SPMD* (Single Program Multiple Data) model. This model implies that every node of the parallel system is running the same program. However, unlike the strict SIMD (Single Instruction Multiple Data) model, all nodes do not have to execute each instruction of the program simultaneously, i.e. in 'lock-step'. So at a given time, each node may execute a different part of the program. Communication between nodes takes place using *communication streams* that are similar to the I/O streams of C++.

6. Discussion

To evaluate the framework several, very different, parallel applications are currently analysed. These applications include: matrix multiplication, Fast Fourier Transform, traveling salesman, and VHDL simulation. For this paper we give a simple example to illustrate the use of the framework in determining the effect of design parameters on the communication performance.

The experiment analyses the (static) throughput behaviour of a 4×4 mesh network with dimension order deterministic routing. A random traffic pattern is used where each node sends 100 messages of 256 data flits (excluding any additional control flits needed for addressing etc.)

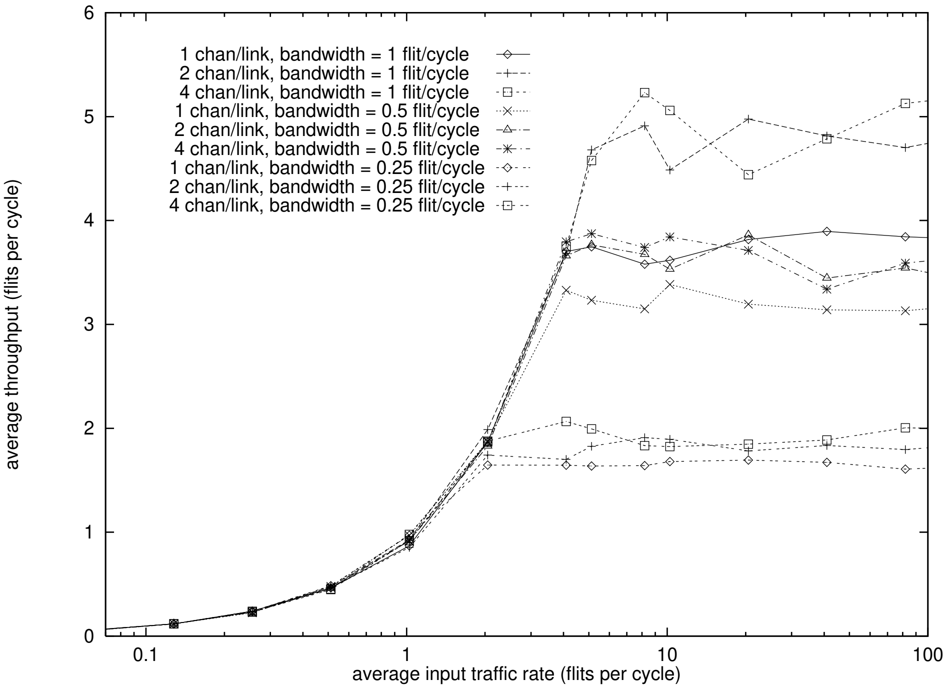

Figure 4. Average network throughput for different network configurations and average input traffic rates.

to other random nodes anywhere in the network. It takes one cycle to send or receive a flit and since this is done by the DP, a node can not send and receive a flit at exactly the same time.

Several simulation runs are done with varying (1, 2, 4) number of channels (and associated channel buffers) per link, varying link bandwidth (1, 0.5, 0.25 flit per cycle) and variable average message injection rate (average input traffic rate)[4]. The resulting throughput is measured and shown in Figure 4.

The results show that up to a certain traffic rate, the resulting throughput keeps up almost linearly. Above that rate the throughput stays constant or even decreases slightly. So each network configuration appears to have an upper throughput limit. As can be expected, increasing the link bandwidth will raise this limit. Also having multiple virtual channels per link improves network throughput, especially with higher link bandwidths, due to less collisions.

Figure 5 shows the average message latencies for the simulations described above. Message latency is defined here as the time it takes between injection of the start of a message in the network until the complete message has arrived at its destination. Obviously, higher average network throughput results in larger average message latencies as the network will become more congested. Networks with higher link bandwidths can have a higher network throughput before this message latency increases significantly. Yet, with increasing network throughput traffic, virtual channels seem to have a negative effect on the message latency. This can be explained by the fact that for a congested network, with increasing number of channels, the available link bandwidth for each channel decreases, thus increasing the latency of the messages it carries. However, as shown in Figure 4, increasing the number of virtual channels yields higher network throughput rates.

[4]Note that this is the amount of traffic that is tried to put into the network, not the actual throughput of the network.

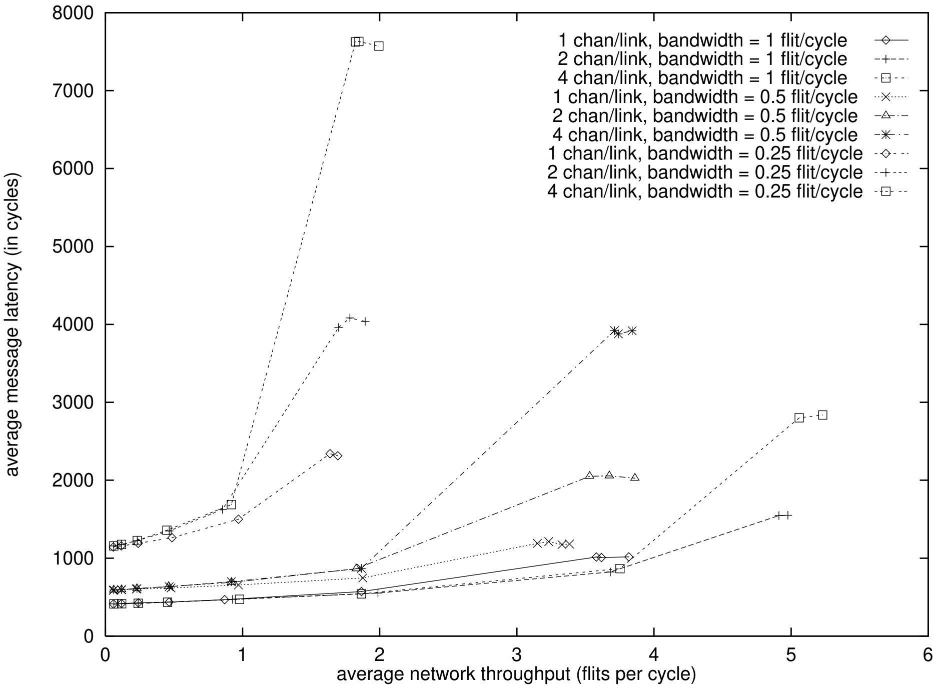

Figure 5. Average message latency for different network configurations and average network throughput rates.

7. Conclusions

A framework is being developed for analysis of parallel applications running on message passing MIMD systems. The goal of the framework is to find the most cost effective communication hardware for a specific application. By analysis of several parallel applications the framework is being evaluated and efforts are made to establish a relation between communication characteristics of analysed applications and the most optimal design parameters values of the corresponding communication hardware.

REFERENCES

1. Geoffrey C. Fox. Lessons from Massively Parallel Applications on Message Passing Computers. In *CompCon '92 Proceedings*, San Francisco, February 1992.
2. Eddy Olk and Henk Corporaal. The OSI Model applied to MIMD Communication Processor Design. In *ParCo 93: Conference on Parallel Computing, Proceedings of the International Conference*, September 1993.
3. Henk Corporaal and Hans (J.M.) Mulder. MOVE: A framework for high-performance processor design. In *Supercomputing-91*, pages 692–701, Albuquerque, November 1991.
4. Eddy Olk. PARSE: Simulation of Message Passing Communication Networks. In *The 27th Annual Simulation Symposium*, pages 115–124, La Jolla, California, U.S.A., April 1994.
5. William J. Dally. *A VLSI Architecture for concurrent Data Structures*. Kluwer Academic Publishers, 1987.
6. H. Corporaal and J.G.E. Olk. A Scalable VLSI MIMD Routing Cell. In *6th Distributed Memory Computing Conference Proceedings, Portland*, pages 654–661. IEEE Computer Society Press, April 1991.