This article has been submitted to the HPCN Europe 1995 conference in Milano, Italy.

MERMAID

Modelling and Evaluation Research in MIMD ArchItecture Design

J. van Brummen A.D. Pimentel T. Papathanassiadis P.M.A. Sloot L.O. Hertzberger

Dept. of Computer Systems
University of Amsterdam
{brummen,pimentel,theopapa}@fwi.uva.nl

Abstract

The Mermaid project focuses on the construction of simulation models for MIMD multicomputers in order to evaluate them and to give estimates of the system's performance. A multi-layered approach was adopted in which three levels can be distinguished. The application level describes application behaviour in an abstract manner, unrelated to any hardware or architecture specifics. Subsequently, the generation level translates these application descriptions to a hardware dependent trace of operations that drives the simulators. And finally, the architecture level consists of the trace-driven architecture simulation models.

Preliminary simulation results show that, despite the high abstraction level at which is simulated, good accuracy can be obtained.

1 Introduction

As the complexity of MIMD multicomputer architectures grows, design options increase and time and budget become more constrained, it is widely recognized that MIMD design should include simulation techniques. With simulation, new architectures can be evaluated and tested long before the actual hardware is built. The evaluation of design options can either focus on functional or performance behaviour. In the scope of the Mermaid project MIMD architectures are simulated from a performance perspective. The Mermaid simulations are performed by means of discrete event simulation and follow the approach of [Muller93].

Two guiding principles were formulated at the beginning of the project. The level of architecture abstraction should allow simulation within reasonable time, preferably avoiding low-level (bus-cycle) emulation. And, architecture choices are important design options which must be simulated without too much remodelling effort.

The (distributed memory) MIMD architectures modelled within Mermaid are the GCel and PowerStone [Langhammer93] architectures of Parsytec. The GCel architecture is based on the Inmos T805 transputer, whereas the initial PowerStone architectures use multiple Motorola PowerPC's for computation and multiple T805 transputers for communication on a node. Since most of the PowerStone architectures are still in design phase, an important research objective of the Mermaid project is to evaluate these architecture designs to be able to give feedback to the designers. To this purpose, the existing GCel architecture is used for validation of the simulation models.

1.1 The Mermaid approach

Performance simulation of MIMD architectures requires an environment in which a continuous development and evaluation of application and architecture models is possible. It requires different levels of abstraction and configuration. Depending on one's research objective the focus of performance simulation might either be at the application level, or at the architecture level, or somewhere in between. It is for this reason that a multi-layered approach was adopted.

Application behaviour is best described in terms unrelated to any hardware or architecture specifics, whereas architectural behaviour is best described in terms of its components and its interactions.

Bringing these two behaviours together to fully cooperate requires an intermediate level where much of our research has taken place. It includes research into computational models, communication models, programming models, programming languages, operating system support, and network support. It bridges the gap between architecture and application, and strives for efficient and optimal architecture design and utilization.

The remainder of this article is structured in the following manner. First, the simulation environment of Mermaid is described serving as a blue-print for the overall project structure. Distinction is made between computation and communication, resulting in two models of different abstraction level describing computational and communication behaviour. Thereafter the architecture simulation models are described as single-node respectively multi-node models. These models have been implemented in the object oriented simulation language Pearl [Muller93]. Then the different levels of application modelling are described. Subsequently, some preliminary simulation results are presented. Finally, conclusions are drawn and future work is discussed.

2 Simulation environment

To evaluate MIMD parallel systems, a parameterized algorithmic model was created. It is capable of supplying the simulator with a (stochastic) trace of events, called *operations*, representing processor activity, memory I/O, and communication message passing. Figure 1 depicts the simulation scheme.

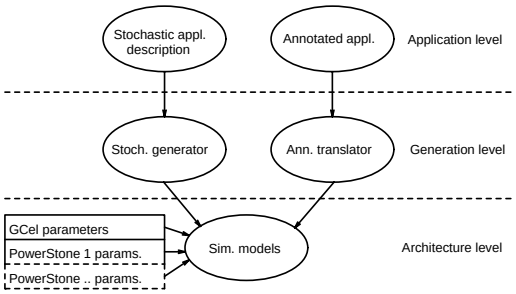


Figure 1: Simulation scheme.

In this scheme, simulation of an application load on an architecture takes place at three different levels:

- Application level: This level contains descriptions and profiles of applications used for input to our simulation model. This is done by either stochastically describing the behaviour of applications using probabilities or by (manually) instrumenting real applications with annotations.

  The application descriptions at this level may range from full-blown parallel programs to small benchmarks to be used to tune and validate the machine parameters of the simulation model.

- Generation level: At this level a trace of operations driving the simulation models is generated from the application descriptions at the application level. The generation process exploits knowledge of the target architecture and runtime model in order to tune the operation traces.

  There are two types of generators, called the *stochastic generator* and the *annotation translator*. The stochastic generator extracts information on the behaviour of applications from the probabilistic application descriptions to produce a "realistic" stochastic trace of operations.

  The annotation translator produces a trace of operations according to the manually inserted annotations at the application level. In comparison with the stochastic generator, this generator can model the application's behaviour more accurately. On the other hand, manually instrumenting applications with annotations is tedious for large applications. In that case, the stochastic generator can be more flexible and easier to use. Therefore, the annotation translator is only used for validation of the simulation model (benchmarks) and for initial application modelling.

  The use of annotations and the annotation translator is further elaborated upon in section 4.1.

- Architecture level: This level consists of (operation) trace-driven simulation models. To allow simulation within reasonable time, these simulation models do not fully emulate the hardware and only limited state information is stored during simulation. For example, the contents of a memory is not modelled and simulated caches only hold addresses (tags), not data.

  Because of the limited state information, the simulators can not provide feedback to the generation process. In other words, there exists an "off-line" relationship between the generation level and the architecture level [Muller93]. This restriction implies that the generators can not make time-dependent decisions, like scheduling decisions in a parallel application. So only operation traces that are not affected by simulation results can be generated.

  Currently, the architecture level includes the GCel model and the PowerStone One model. Every model has a set of machine parameters that have been calibrated by either published information or benchmarking.

2.1 Restrictive to the SPMD model

As a starting point for our research we have chosen the SPMD (Single-Program, Multiple-Data) programming model [Gupta91] because it is considered to be a very

realistic model when programming a massively parallel MIMD platform. In this model every MIMD node executes a single common program, exploiting implicit parallelism by focussing on its share of the multiple data.

Applications adhering to the SPMD programming model mainly exhibit a coarse-grain interaction of computation and communication. The large parts of computation within these applications are interchanged with periods of communication.

Besides the SPMD programming model, support for explicit parallelism by means of multi-threading [Birrell89] is another established aspect in parallel programming. However, multi-threading does not mix with the SPMD paradigm: threads may be scheduled to run in parallel on different processors, thereby disobeying the SPMD programming model.

An additional problem of multi-threading is that it is difficult to model. The reason for this is that multi-threading becomes important from a performance point of view when latencies can be hidden. This makes the thread scheduling depend on the hardware's resources and latencies, which implies that the application's operation trace would be affected by the simulation results. This would invalidate our approach of an off-line generated application trace.

Therefore, it was decided to limit the application behaviour at the application level to the SPMD programming model. Explicit parallelism like multi-threading of applications is not supported.

2.2 Validation

The purpose of the simulation is to forecast performance figures of a number of successive PowerStone architectures for a characteristic application load. The machines are under development and are not, or only very limited, available for the purpose of validation. Full comparison of simulation and execution results can only be done for the existing GCel architecture, and in retrospect for the successive PowerStone machines after completion.

The purpose of reflecting on validation must be a continuous effort during our simulation work as greater accuracy and wider scope is strived for. A simulation of a parallel application running on a MIMD architecture with a lot of interacting hardware components on one node and the node configured in a certain communication topology, is a difficult task. It is for this reason that a process of stepwise refinement is preferred. With this approach a degree of accuracy can be established for a growing subset of operations and a growing complexity of architecture.

When modelling MIMD architectures using operation traces to drive the simulators, three areas can be distinguished where validation is necessary:

- Does stochastic generation of a synthetic trace of operations resemble the behaviour of real applications?
- Is the translation from annotations to operations accurate for a given target processor and runtime model?
- Does the architecture model resemble the latencies of the real operations under different loads of operations?

To verify the accuracy of the stochastic generation, the performance critical parts of a typical application can be adapted to generate an exact algorithmic operation trace of the application that exactly follows its data and instruction access patterns. From this trace of operations the amount and locality of the application's local and global data access, its instruction access, and its communication behaviour is quantified. The resulting quantification can then be used during or after stochastic generation as a semantic accuracy check.

On the other hand, the stochastic generator does not always have to strictly operate within the range of application behaviour, as there is the additional interest in performance degradation and its successive bottle-neck analysis under very extreme architecture loads.

Validating the translation of annotations to operations can be done by a comparison of generated assembly code and generated simulation operations, if a compiler for the target processor is available. Specific compiler optimizations can be detected, which may result in tuning the annotations or even in adjusting the translator.

For the validation of the architecture models a suite of annotated benchmarks producing high quality operation traces can be simulated, after which the results are compared with executed performance results. These benchmarks can be simple and only concentrate on rudimentary architecture operations. Once these operations have been correctly calibrated, validation can be extended to cover larger and more complicated benchmarks or applications.

2.3 Level of simulation

Trace-driven simulation can be performed at different abstraction levels. Depending on the research objective, the operation traces can either be simulated at the level of bus-cycles, instructions, basic blocks or even tasks [Hartel93]. Accuracy improves by descending in abstraction level, but this results in an increased computational intensity.

To arrive at the proper abstraction level of architecture simulation and to establish its interface of supported operations the following ground rules were adhered to:

- The operations must abstract from the nitty-gritty details of the processor architecture's instruction sets.
- The operations must be abstract enough to interface to the architecture model of the GCel as well the architecture models of the different PowerStone architectures.
- It is necessary that the operations abstract from programming languages and computational models, and are capable of representing the applications at the algorithmic level.

2.3.1 Computation versus communication

Because of the coarse-grained interaction of communication and computation within SPMD applications and the fact that latencies of computation and communication are so diverse in terms of architecture modelling, simulation of application behaviour was split into two different models. Both of the models define their own set of operations. The first model is typical for the application's computational behaviour and models at an abstraction level in between bus-cycles and instructions. This model drives the second model by providing it with information about computation at task level. The second model is typical for the application's communication and synchronization behaviour. This model operates at task level for computation and at instruction level for communication. It implements the computational tasks, derived from the computational model, as delays between communication requests.

This approach results in a *hybrid model* which is only valid if the performance of computation and communication do not influence each other. This is a reasonable assumption for state-of-the-art SPMD programming in general.

2.3.2 Basic simulation operations

Computational instructions of both the Inmos T805 and the PowerPC processors, the computational building blocks of respectively the GCel and PowerStone architectures, do not modify memory. Memory access is only possible through explicit loads and stores, or through instruction fetching. Thus, it makes sense to define the architecture of our simulation model as a RISC load-store architecture [Hennessy90] with memory-to-register and register-to-memory transfers.

The operations available to the computational simulation model are shown in Table 1.

load(memory-type, address) *store(memory-type, address)* *load([f]constant)*
add(arithmetic-type) *sub(arithmetic-type)* *mul(arithmetic-type)* *div(arithmetic-type)* *cmp(arithmetic-type)* ...
ifetch(address) *branch*

Table 1: Computational operations

The operations can be subdivided into three categories. The first category consists of operations for transferring data between the architecture's registers and memory hierarchy. Parameters of these operations indicate the type of memory reference and, if appropriate, the memory location.

The second category of operations are arithmetic functions that solely operate on registers. The associated parameter indicates the type on which the function should be performed. This can be integer, single precision or double precision floating point. Besides the arithmetic operations shown in Table 1 there are some additional geometric operations.

Finally, the third category of operations are associated with instruction fetching. With the *ifetch* operation, an instruction fetch from memory location *address* can be modelled.

As the simulation will not emulate the operations, the simulator will not be aware of loops and branches. The application trace generator evaluates loop and branch-conditions, and generates the operation trace for the invocated control flow. This implies that every invocation of a loop body is individually traced and leads to recurring addresses of instruction fetches.

The *branch* operation models the target processor's branching latency and triggers its branch prediction scheme if available.

The basic communication operations that act as input for the communication model relate to the functionality that is provided by the operating system running on both the GCel and PowerStone machines. This operating system, called PARIX [Parsytec92], uses *virtual links* as the basic means of communication.

send(message-size, destination) *recv(source)*
asend(message-size, destination) *arecv(source)*
compute(duration)

Table 2: Communication operations

The operations available for the communication model are listed in Table 2. The *source* and *destination* identifications are based on the PARIX numbering scheme of its two-dimensional grid communication architecture.

There are both operations for synchronous and asynchronous communication. Currently, the asynchronous communication operations are not yet implemented. Additional to the support for communication, the computational requirements within the communication model are modelled at task level by means of the *compute* operation.

3 Architecture modelling

As application behaviour divides in computational behaviour and communication behaviour, two different architecture models of the MIMD platform are required.

The architecture model for computation involves one single-node of the MIMD platform with its processors, caches, bus, and memory. Simulation will be performed in between instruction and bus-cycle level.

The architecture model for communication implements the MIMD node in a very abstract manner as just a processor and a router. The nodes are linked together resulting in a multi-node model to reflect the platform's interconnection scheme. The router corresponds with the platform's physical communication topology, which is a two-dimensional $N \times N$ grid for the Parsytec architectures. Simulation of computational requirements is performed at task level, while communication is simulated at the lower instruction level.

3.1 Single-node architecture

The architecture models for the simulation of computation are generic in the sense that they can be parameterized to represent several node architectures. The model is shown in Figure 2.

The base node model for computation is capable of modelling both the GCel and the PowerStone node architectures without making dramatic changes. It has been configured for the GCel which is based on the T805 transputer, and for the PowerStone One which is based on a

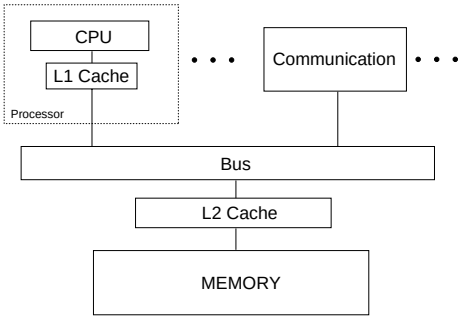


Figure 2: The base single-node model.

combination of Motorola PowerPC 601 and T805 transputer technology. The same model will be used for future PowerStone architectures.

The CPU component simulates the CPU of the node architecture. It supports an operation set, as described in section 2.3.2, which is more abstract than the real machine instruction set. The operations are associated with fixed execution latencies. The CPU must be configured with an operation set, operation latencies in clock cycles, the clock speed, the on-chip registers, and specific details on pipelined and super scalar processing.

The L1-cache simulates the first level cache of the memory hierarchy. It implements a cache coherency protocol if multiple CPUs are configured on one node. Both the configurations as a separated instruction/data cache and a unified cache are possible. Other parameters are the cache size, its associativity, its block replacement strategy, its write strategy, and the latencies associated with cache hits, misses, and replacements. Additionally, instead of cache it can also be modelled as on-chip static memory.

The bus simulates the main bus of the node architecture as a simple forwarding scheme implementing bus arbitration upon multiple access. Depending on node configuration it is placed either between L1-cache and L2-cache, or between CPU and L1-cache. It is parameterized with bus-width, bus clock-rate, and arbitration details.

The L2-cache simulates the second level cache of the memory hierarchy if available in the architecture. It supports the cache coherency protocol of the L1-cache if required. It can be configured either as separated instruction and data cache, as unified cache, or as stream cache [MacroTek93]. Parameters include cache size, associativity, block replacement or stream pre-fetching, and the latencies associated with hits, misses, and replacements or pre-fetches.

The memory simulates a simple DRAM memory. It is parameterized with memory size, memory refresh rate,

and memory access latencies.

The communication component models the processing of communication requests. The computational node behaviour as defined in previous sections does not involve this component. For this reason the component is currently ignored. However, it could become important in the future at the level of board design when different implementation choices for communication processing must be considered. This is especially true for the PowerStone architectures with its diversity of communication hardware components.

3.2 Multi-node architecture

The communication load of the SPMD application results in a communication trace for each node of the MIMD platform. The main objectives of this approach are:

- It allows for all kinds of load balancing scenario's by means of different *compute* operations on different nodes.
- It allows for all kinds of synchronization scenario's because every node can communicate with every other node.
- It models the routing latency of non-neighbourhood communication.
- It models the possible impact of routing overhead on the node's computational processing.

The required architecture model to simulate these traces has to model the communication hardware and software at each MIMD node. The node model consists of a processor and a router. It connects to maximal four full-duplex communication links. Processor and router don't have to be implemented by different hardware components. The GCel uses its T805 transputer both for computation and routing, whereas the PowerStone One architecture uses two PowerPC's for computing and four transputers for routing. The two PowerPC's on one node in the PowerStone One can either be modelled as two separate nodes or as one node in which the multiple PowerPC's are abstracted away. The node model is shown in Figure 3.

The nodes are configured as a two-dimensional $N \times N$ grid by connecting all neighbouring nodes with the full-duplex links. The processor object of the node reads the incoming operation trace, processes the *compute* operations, and dispatches the communication requests to the router object. The router object dispatches incoming communication requests from the processor to multiple packets if required. Further, the router object routes the resulting and all other incoming packets

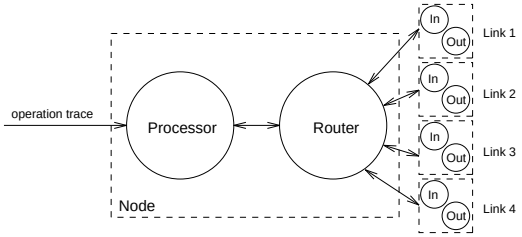


Figure 3: The base multi-node model.

through the two-dimensional grid using the deterministic XY-routing strategy [Ni91] with store-and-forward switching. The router object can be replaced to implement the deterministic geometric routing scheme of [Badouel91]. Router replacement allows the model to be adapted for a range of other routing strategies like those of [Annot87, Mooij89, Schwiebert93, Ni91] in order to evaluate their performance impact.

The platform's routing strategy is just one example of architecture behaviour which is implemented in software and which must be taken care of. The way in which messages are split-up in different packets and whether and how messages are synchronously or asynchronously sent. These are all examples of implementation defined behaviour of the MIMD platform's message passing layer.

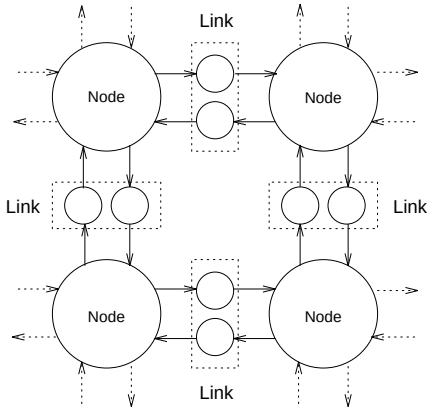


Figure 4: The base topology model for communication.

The communication topology is automatically generated by configuring $N \times N$ nodes and connecting the nodes with the appropriate uni-directional connections. Pairs of two uni-directional connections of opposite direction between two nodes implement the node's full-duplex communication links. The configuration is demonstrated in Figure 4.

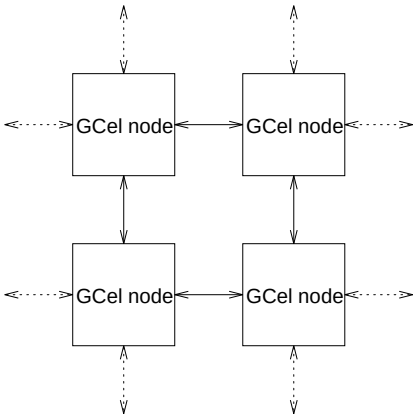


Figure 5: The GCel interconnections.

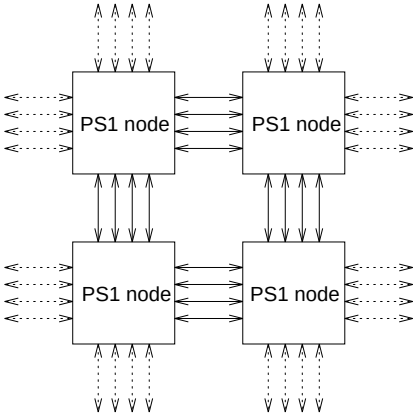


Figure 6: The PowerStone One interconnections.

3.2.1 GCel grid architecture

The GCel communication architecture consists of linking T805 transputers in a two-dimensional $N \times N$ grid. The T805 transputer is responsible for both computation and communication which implies that routing overhead has its impact on computation. Figure 5 shows the single transputer links between the nodes. The GCel communication model is parameterized with the packet size, and with the T805's link setup, link transmission, packet routing, and packet store-and-forward latencies.

3.2.2 PowerStone One grid architecture

Figure 6 shows the PowerStone One communication architecture with its two dimensional $N \times N$ grid with four T805 transputer links between neighbouring nodes. PARIX regards the four T805 transputer links as four separate $N \times N$ grids. Messages are split-up in packets which are divided over the four grids. The packet protocol on every grid layer remains the same. In this way PARIX maintains its transputer implementation model while increasing communication capacity.

The PowerStone One communication model is parameterized with the packet size, with the T805's link setup and packet routing latency, but with a factor 4 downscaled T805 transmission and store-and-forward latency because of the four T805's working in parallel.

4 Application modelling

Most of the SPMD programs that run on MIMD platforms are scientific applications dealing with large multi-dimensional floating-point arrays. Modelling these applications by means of operation traces implies that these traces will have to contain the application's characteristics.

Generation of operation traces using annotation techniques and exploitation of knowledge of the target architecture and runtime model should automatically lead to an accurate view of application behaviour. However, the use of the stochastic generator for producing application loads is less trivial. In order to produce high quality stochastic operation traces, care must be taken that the generated traces reflect the application behaviour with respect to:

- the amount and locality of instruction-fetches representing the application's performance critical inner loops,
- the mix of load, store, and computational operations to represent the application's characteristic operation mix,
- the amount and the locality of data loads and stores on the stack representing scratch values of intermediate sub-expression and function results,
- the amount and the locality of data loads and stores in global memory with respect to the application's single- or multi-dimensional global or static floating-point arrays,
- the amount and locality of communication data and the locality of communicating processes with respect to processor topology,
- synchronization of data consuming processes and data producing processes.

At the current stage of the project trace generation is only possible through annotation of applications. Formalisms that express application behaviour

using stochastic methods still need to be devised. Consequently, the stochastic generator has not yet been designed and implemented.

The remainder of this section describes the methods used for application modelling by means of annotations. Additionally, the idea of an SPMD communication model is presented. Eventually, this model should be the basis of all application's communication behaviour modelling within the Mermaid project.

4.1 Application modelling by means of annotations

Instrumentation of applications to generate operation traces can be performed automatically during compilation of the application. Such an automatic instrumentation of code with annotations is convenient. In fact, doing this manually is a tedious job which can be easily lead to mistakes. However, in this simulation project automatic instrumentation was not feasible. The reason for this was two-fold: compiler sources were not available for adaption, and more importantly the project targets for simulation as a means of evaluating future processor choices in an early stage without silicon or compiler being yet available. In this project this was typically the case with the PowerPC processor.

4.1.1 Modelling computation

It was decided to restrict modelling of the computational behaviour of an application to just those parts of the application being computationally intensive. These parts are often referred to as the *kernel functions* of the application. They are an important ingredient to the overall performance estimate of the application. To achieve speed-ups they are carefully optimized to avoid unnecessary overhead for function calls, loops, computations, and memory references.

At the application level, these kernel functions are manually instrumented with annotations that follow the original control flow of the function and represents the function's memory and computational behaviour. The instrumentation is such that the kernel functions can either be compiled for execution or for trace generation.

At the generation level, the annotations are translated to a representation of basic operations. This translation is guided by a number of principles which must be carefully adhered to. Every function variable used in a kernel function has an entry in a table of function variable descriptors that determines: whether the variable is a function argument or a function automatic, the offset in the stack-frame where the variable resides, whether the variable is placed in a register or not, the type of the variable. Register re-assignment of function variables can be modelled by dynamically updating their descriptors. When, for example, an annotation indicates that a variable should be loaded, the generator will translate this into one or more instruction fetches and appropriate memory operations according the runtime model and addressing capabilities of the targeted processor

4.1.2 Modelling communication

Annotations describing communication behaviour at application level directly map onto the PARIX virtual-link communication primitives. At the generation level, the annotations are translated to the appropriate communication operations (listed in Table 2). These communication operations reflect the architecture's underlying physical communication topology with its communication processors, interconnections and routing strategies.

Currently, only communication defined on the physical communication topology can be modelled. Support for modelling communication within application-defined communication structures, called virtual topologies [Simon93, Rottger94], is not yet available. In order to include such support, an extra translation step is required that translates the virtual communication requests to physical communication requests. This translation must be guided by a pre-established mapping of the virtual communication topology on the platform's physical communication topology. Efficient mapping of virtual topologies is a topic of on-going research [Rottger94]. For this reason, it might be more flexible to incorporate the mapping of a virtual topology as a configuration parameter at generation level rather than including the mapping algorithms.

4.2 The SPMD communication model

Modelling of communication that is strictly defined on the architecture's physical communication topology, as described in the previous section, still reflects all of the underlying hardware characteristics. By modelling virtual communication topologies, these architectural details can more or less be hidden at the application level. In this section, we will go one step further and present an SPMD communication model that abstracts from all kinds of explicit communication at the application level. This model will be used as a basis for future development of formalisms expressing application's communication behaviour, both for the stochastic and annotation approaches.

In the SPMD communication model, communication behaviour of an application is modelled at the applica-

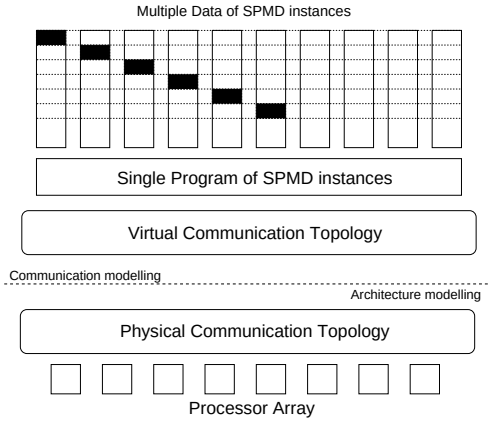


Figure 7: The SPMD Communication Model.

tion level in the form of an abstract description of data requests without any details on communication primitives or underlying communication architecture. The generation level first translates these abstract requests to communication operations for a virtual communication topology. After that the virtual communication requests are further translated to the basic communication operations at architecture level.

Figure 7 shows all three levels at which communication is expressed: the SPMD's Multiple Data and Single Program represent the application level, the virtual communication topology represents the generation level, and the physical communication topology and communication processors represent the architecture level.

4.2.1 Communication at application level

At application level the model regards communication behaviour as an implicit result of accessing distributed data in one shared address space. This approach is similar to that originally proposed by [Kennedy88] and later adopted by the HPF-FORTRAN initiative [HPF-Forum93]. The parallel SPMD program consists of multiple SPMD instances which will be called SPMD processes. The SPMD processes share a single program which must be accessible from every processor. The shared address space encompasses data and instructions, and of which the data segment represents virtually shared data. The virtually shared data incorporates the SPMD application's distributed data. Distributed data can either be allocated with the process itself or with one of the other processes. Access to distributed data will result either in a memory request or in a communication request depending on its distribution scheme.

Figure 7 demonstrates the distribution of SPMD data (black areas) filling up slots of the SPMD shared data address space. Distributing SPMD data equally over multiple SPMD processes is not necessarily the case and will depend on the balance of required computational power versus the amount of processed data. If certain data areas are very computationally intensive the SPMD programmer could decide to split these areas over multiple processes producing a more balanced program behaviour and higher throughput.

There are a number of issues that must be carefully considered when modelling the communication behaviour of an application with the SPMD communication model:

- Access of distributed data needs to be synchronized to avoid data hazards like RAW (*read after write*), WAR (*write after read*), and WAW (*write after write*) for the parallel executing SPMD processes; these hazards are similar to the data hazards with instruction pipelining [Hennessy90].

- Some computational models or applications implement the *owner-computes-rule* which specifies that only one SPMD process owning the *left-hand-side* of an expression computes its corresponding *right-hand-side* and updates the *left-hand-side*'s data item [Kennedy88].

- The *compute*-latencies must be derived from the computational modelling, and must be embedded in the communication model between the communication requests.

- Rather than instrumenting the application to generate a communication trace it might be more efficient and flexible to model its communication behaviour stochastically.

4.2.2 Communication at generation level

At generation level the model translates the SPMD data requests to communication requests within a pre-established virtual communication topology. At the level of virtual communication the connectivity of the unconstrained point-to-point SPMD data request is reduced to just those connections available within the virtual topology. This means that virtual topology routing must be introduced to service the 'unconnected' point-to-point communication requests. Such virtual topology routing can be implemented as a sequence of multiple virtual topology communication requests.

After generating the virtual communication requests the generation level takes care of translating them further to physical communication requests. This translation step is equal to the one described in section 4.1.2.

5 Experiments

Currently, the Mermaid project has come to a stage in which the GCel models are to be validated. This can be done by comparing the simulation results with real execution results on the GCel machine. After the validation stage, experiments can be extended to include PowerStone One and future PowerStone architecture simulations.

This section describes the results that were obtained by simulating the architecture behaviour of the GCel machine for a number of benchmarks. The operation traces were obtained by instrumenting the benchmarks with annotations. The GCel platform used for validation belongs to the IC^3A (Interdisciplinary Center for Complex Computer facilities Amsterdam) and consists of 512 T805 transputers.

Simulation results of architecture loads due to computation relate to a single node of the GCel platform and simulation results of architecture loads due to communication relate to multiple nodes of the GCel.

5.1 Computation

The computational benchmarks that have been performed are:

- Ddot, a double precision innerproduct
- Ddot4, a loop unrolled version of Ddot
- Daxpy, a double precision vector update
- The kernel function of the Elastic Light Scattering simulation of [Hoekstra94] called dipole.

The ddot, ddot4 and daxpy benchmarks are scaled to an array size N of 8192 doubles by performing $\frac{8192}{N}$ outer iterations in order to maintain precision at smaller array sizes.

The measurements of the benchmarks are shown in Figure 8. The three figures for the ddot, ddot4, and daxpy measurements contain graphs for results of GCel execution and simulation. The fourth graph shows the error margin of simulation compared to execution. And the results of the dipole benchmark are shown in Table 3.

GCel execution	GCel simulation	Error %
13664	13144	3.8

Table 3: Dipole results in micro seconds.

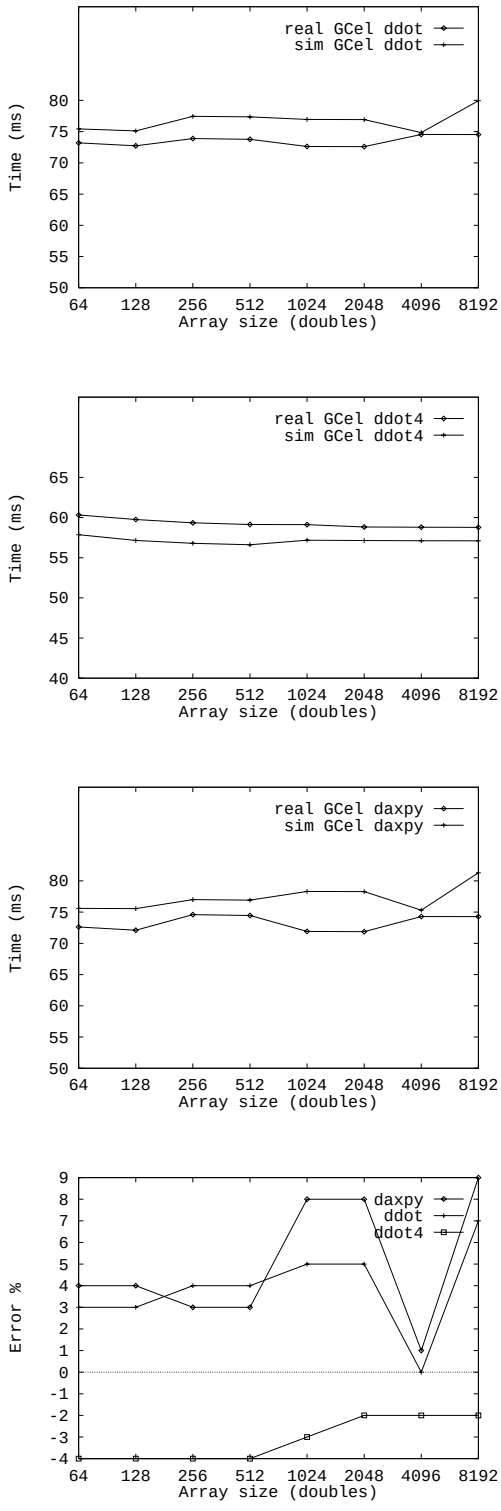


Figure 8: Results for ddot, ddot4, and daxpy.

The simulation model behaves reasonable well for these benchmarks as can be seen from the error margins. Simulation of the more complex dipole benchmark even gives a proper estimation.

The dip in the error margin graph of Figure 8 at an array size of 4096 is probably caused by the modelled (paged) DRAM, since we measured an increased number of page-hits at this particular size.

5.2 Communication

The communication benchmarks that have been performed are parameterized for a fixed message size of S bytes and are scalable to any amount of processors N. All benchmarks use synchronous communication.

One benchmark, called ping-pong, was used for fine-grain validation of the communication model. It sends a message of size S bytes from one node to another node after which the message is sent back again. So this benchmark measures the transmission time of single messages. The other three benchmarks are somewhat more sophisticated and are used for stress-testing the communication model. These benchmarks iterate over a pre-defined number of time slots and generate:

- A synchronized all-to-all communication load in which every processor communicates with all other processors resulting in $N * (N-1)$ overall communication requests of S bytes per time slot.
- A synchronized point-to-point communication load in which every processor communicates with one fixed partner $N-1$ times resulting in $N * (N-1)$ overall communication requests of S bytes per time slot. The partner tuples are formed by the mapping $(p, ((\frac{N}{2}) + p) \bmod N)$ which result in approximately equal-distanced routing paths. This benchmark will be referred to as *equal-distance communication* from now on.
- A synchronized point-to-point communication load similar to that of the previous load but with a different partner mapping $(p, (N-1) - p)$ which results in a much bigger variety of routing paths ranging from minimal 1 or 2 to maximal $2 * (N-1)$ hops. This benchmark will be referred to as *unequal-distance communication* from now on.

The number of time slots starts at 600 (at a message size of 64 bytes) and is scaled down for larger message sizes according the formula: *#time slots* $= \lfloor 200 * \frac{MessageSize}{128} \rfloor$. Further, the communication benchmarks were limited to at most 64 processors to avoid hazards like excessive simulation time and excessive disk storage for the trace files.

The results of the ping-pong benchmark are depicted in Figure 9. The graph displays the error margin between execution and simulation for several hop-counts (the distance of the message destination). With an error margin that does not exceed the 7 percent, one can conclude that the communication model is fairly accurate for simple single-message modelling. The higher error margin for small messages demonstrates that smaller messages experience greater influence of implementation dependent optimizations.

The measurements of the other three benchmarks are shown in Figure 10 for all-to-all communication, in Figure 11 for equal-distance point-to-point communication, and in Figure 12 for unequal-distance point-to-point communication. Each figure contains measurements of the same benchmark for three different grid sizes. Measurements for a specific grid combine the results for GCel execution and simulation in one graph. The decreasing execution (and simulation) time for increasing message sizes is caused by the down-scaling, as mentioned earlier. The fourth graph shows the error margin of simulation compared to execution.

Communication performed by these three benchmarks are examples of extreme cases, resulting in higher error margins compared to the simple single-message benchmark. However, it remains the question whether real SPMD applications exhibit such extreme communication behaviour.

Figure 10 shows that simulation of the all-to-all communication benchmark is reasonable accurate. The smooth paths of the error margin curves emphasize the regular routing load this benchmark produces.

The error margin graphs of the equal and unequal-distance point-to-point benchmarks (Figures 11 and 12) show a much more whimsical behaviour. This can be explained by the more non-uniform routing loads these two benchmarks produce. From Figure 11 can also be

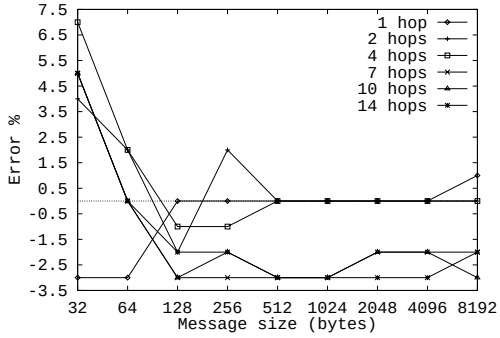
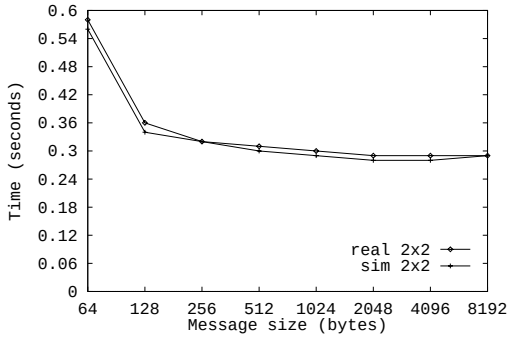
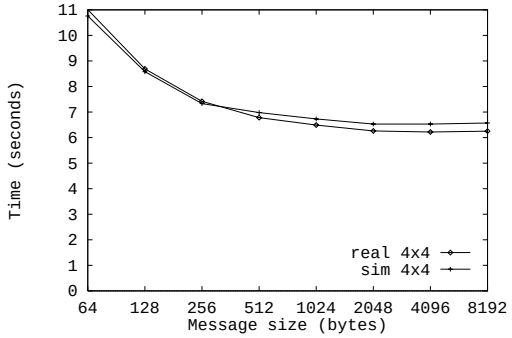
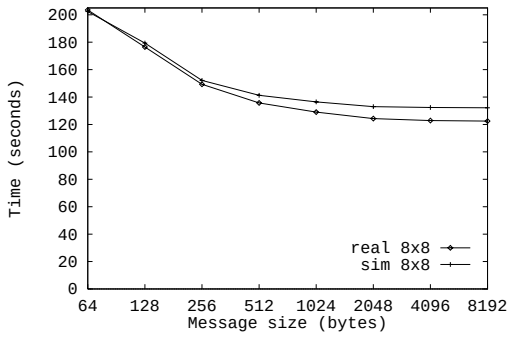


Figure 9: Ping-pong benchmark.







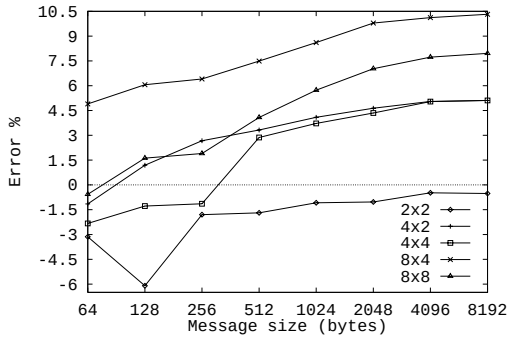
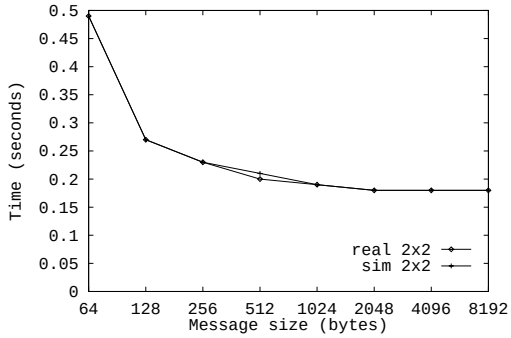
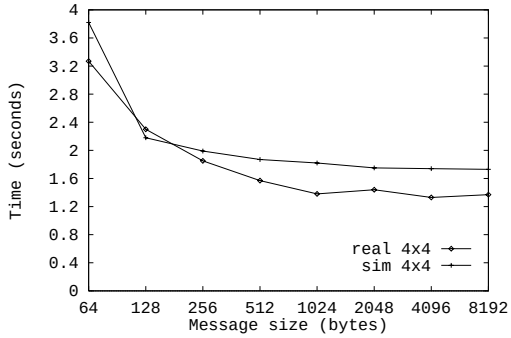
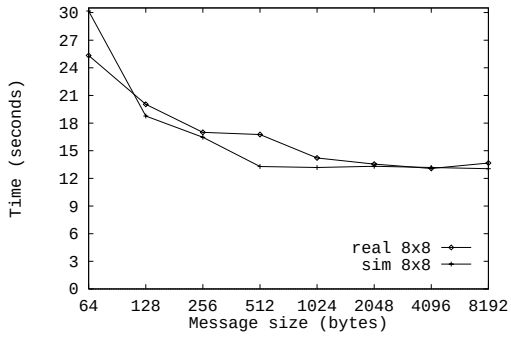


Figure 10: All-to-all communication.







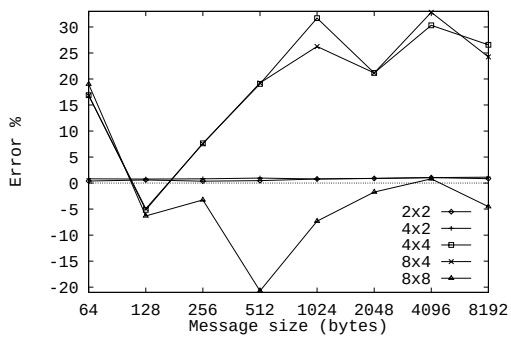


Figure 11: Equal-distance point-to-point communication.

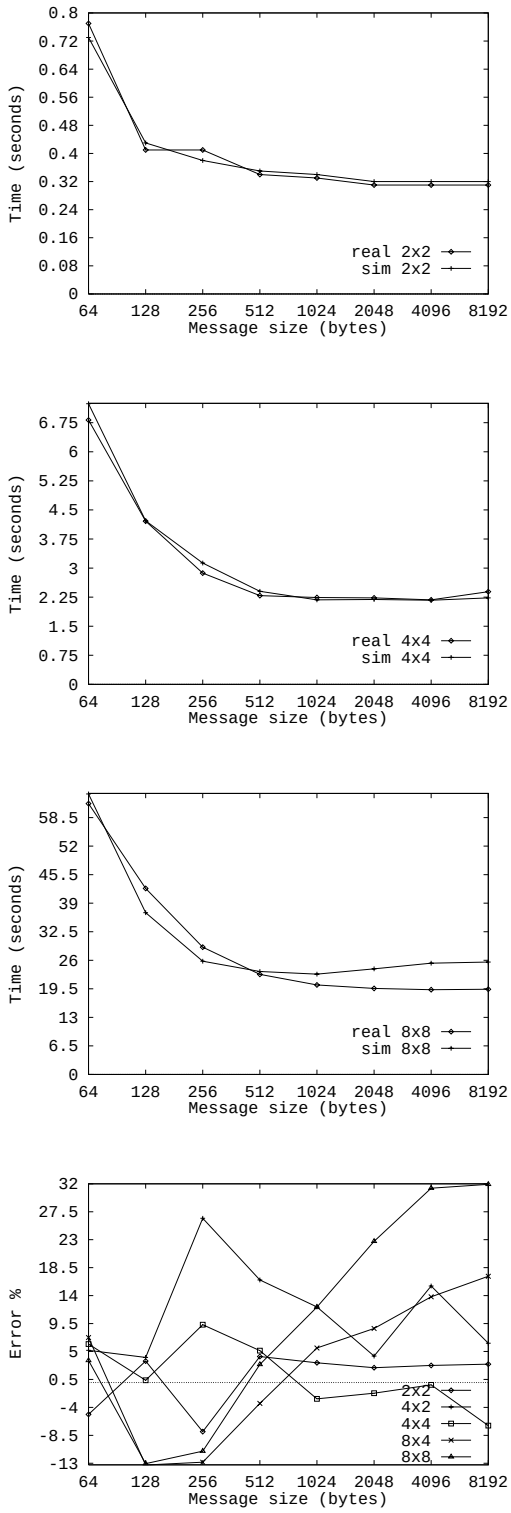


Figure 12: Unequal-distance point-to-point communication.

seen that the equal-distance benchmark only depends on the used Y-dimension of the communication network. This is because the communication in this benchmark is only performed along the Y-axis.

6 Conclusions

Within the Mermaid project, a frame-work for MIMD architecture modelling and simulation with the purpose of performance evaluation has been developed. The off-line generated operation traces that drive the simulators were kept as abstract as possible to allow simulation within reasonable time. Additionally, there has been strived for high quality operation traces, since representative application loads are paramount for architecture simulation and successive evaluation of results.

The validation of the computational and communication architecture models by means of benchmarks demonstrated that

- careful annotation of computational kernel functions can produce representative application loads that lead to accurate simulation.
- implementation issues cannot be ignored. An illustration for this is the inefficient memory behaviour imposed by the T805 transputer runtime model.
- the multi-node simulations at the level of packet transmission for communication enable evaluation of different message passing and routing strategies.

Validation of the architecture models is a continuous effort. The multi-node communication experiments generating more complex loads show a greater diversity of error margins. During the project, results have improved considerably and the project is currently in a state where more random and irregular behaviour can be validated.

6.1 Future work

The work in the Mermaid project can proceed in many ways. A description of future research and related development is given at the application level, generation level and architecture level of the project.

At the application level, a formalization must be constructed to stochastically describe application behaviour. For communication, this formalism should be based upon the SPMD communication model with its abstract data requests.

Further, research can be performed on different strategies for data replication and data migration that can be expressed by an extension of the SPMD communication

model. These strategies might have an interesting impact on the simulation of the generated multi-node architecture load. This work requires that the formalisms and associated tools of the SPMD communication model are available, and that the generation level contains the further translation to physical communication requests.

At the generation level, the stochastic generator must be designed using the formalisms constructed at the application level as guideline. This work would also require the specification of a number of input samples, and the validation of their resulted generated stochastic operation stream.

Concerning the SPMD communication model, the translation of SPMD implicit data requests to explicit communication requests, and from virtual communication requests to physical communication requests requires the research of formalisms in which these behaviours can be expressed. The translation should also handle synchronization requests.

The work at architecture level can proceed along one of more of the following routes. The single-node architecture models could be extended to include more components to simulate the implementation choices for communication request processing. Besides this, extensions could also be made to improve the support for design option evaluation of memory hierarchies.

Finally, overall system performance evaluation of the future PowerStone architectures will be an important ingredient for future research. This work extends in a general sense on the PowerStone One simulations, and takes the current PowerStone Two and PowerStone Three designs as input.

Acknowledgements

We would like to thank Marcel Beemster for his comments on draft versions of the article and Alfons Hoekstra for his contributions with respect to the Elastic Light Scattering benchmark. We also want to express our gratitude to Parsytec for their support.

References

[Annot87] J. K. Annot and R. A. H. van Twist. *A Novel Deadlock Free and Starvation Free Packet switching Communication Processor. Proceedings PARLE 87,* 1,2, 1987.

[Badouel91] D. Badouel, C. A. Wüthrich, and E. L. Fiume. *Routing Strategies and Message Contention on Low-dimensional Interconnection Networks.* Technical report, Comp. System Research Institute, University of Toronto, 1991.

[Birrell89] A. D. Birrell. *An Introduction to Programming with Threads.* Technical report, Digital Systems Research Center, 1989.

[Gupta91] R. Gupta. *SPMD Execution of Programs with Dynamic Data Structures on Distributed Memory Machines.* Technical report, Department of Computer Science, University of Pittsburgh, 1991.

[Hartel93] P. Hartel, R. F. Hofman, K. G. Langendoen, H. L. Muller, W. G. Vree, and L. Hertzberger. *A toolkit for parallel functional programming.* Technical report, Dept. of Comp. Sys, Univ. of Amsterdam, 1993.

[Hennessy90] J. L. Hennessy and D. A. Patterson. *Computer Architecture A Quantitive Approach.* Morgan Kaufmann Publishers, Inc., San Mateo, California, 1990.

[Hoekstra94] A. G. Hoekstra. *Computer Simulations of Elastic Light Scattering.* PhD thesis, Dept. of Comp. Sys, Univ. of Amsterdam, 1994.

[HPF-Forum93] HPF-Forum. *High Performance Fortran Language Specification, version 1.0 CRPC-TR92225.* Technical report, Center for Research on Parallel Computation, Rice University, Houston, TX, 1992 (revised May 1993).

[Kennedy88] K. Kennedy and D. Callahan. *Compiling Programs for Distributed-Memory Multiprocessors. The Journal of Supercomputing,* vol. 2, pp. 151-169, 1988.

[Langhammer93] F. Langhammer. *The PowerStone Project, Version 1.2.* Technical report, PowerStone Parallelrechner GmbH, Herzogenrath, Germany, 1993.

[MacroTek93] MacroTek. *High Bandwith Resource Interface Controller Chip Set for '601 Based Systems.* Technical report, MacroTek GmbH, Dortmund, Germany, 1993.

[Mooij89] W. G. P. Mooij. *Packet Switched Communication Networks for Multi-Processor Systems.* PhD thesis, Dept. of Comp. Sys, Univ. of Amsterdam, 1989.

[Muller93] H. L. Muller. *Simulating computer architectures.* PhD thesis, Dept. of Comp. Sys, Univ. of Amsterdam, 1993.

[Ni91] L. M. Ni and P. K. McKinley. *A Survey of Routing Techniques in Wormhole Networks.* Technical report MSU-CPS-ACS-46, Dept. of Comp. Sc., Michigan State University, 1991.

[Parsytec92] Parsytec. *PARIX Documentation for Release 1.1.* Technical report, Parsytec Computers GmbH, Aachen, Germany, 1992.

[Rottger94] M. Rottger, U.-P. Schroeder, and J. Simon. *Virtual Topology Library for PARIX.* Technical report, Paderborn Center for Parallel Computing (PC^2), Department of Mathematics and Computer Science, University of Paderborn, Germany, 1994.

[Schwiebert93] L. Schwiebert and D. N. Jayasimha. *Optimal Fully Adaptive Wormhole Routing for Meshes.* Technical report, Dept. of Comp. and Inf. Sc., Ohio State University, 1993.

[Simon93] J. Simon. *Benutzung virtueller Topologien unter PARIX, TR-006-93.* Technical report, Paderborn Center for Parallel Computing (PC^2), Department of Mathematics and Computer Science, University of Paderborn, Germany, 1993.