A Communication Model for Small Messages with InfiniBand

Torsten Hoefler and Wolfgang Rehm*
{htor, rehm}@informatik.tu-chemnitz.de
Chemnitz University of Technology - Chair of Computer Architecture
Strasse der Nationen 62, 09111 Chemnitz

Abstract

Designing new and optimal algorithms for a specific architecture requires accurate modelling of this architecture. This is especially needed to choose one out of different solutions for the same problem or to proof a lower bound to a problem. Assumed that the model is highly accurate, a given algorithm can be seen as optimal solution if it reaches the lower bound. Therefore the accuracy of a model is extremely important for algorithmic design. A detailed model can also help to understand the architectural details and their influence on the running time of different solutions and it can be used to derive better algorithms for a given problem. This work introduces some architectural specialities of the InfiniBand network and shows that most widely used models introduce inaccuracies for sending small messages with InfiniBand. Therefore a comparative model analysis is performed to find the most accurate model for InfiniBand. Basing on this analysis and a description of the architectural specialities of InfiniBand, a new, more accurate but also much complexer model called LoP is deduced from the LogP which can be used to assess the running time of different algorithms. The newly developed model can be used to find lower bounds for algorithmic problems and to enhance several algorithms.

1 Introduction

Models for parallel programming are often used to develop and optimize time critical sections of algorithms for parallel systems. These models should reflect all relevant parts of real-life-systems for algorithmic design. Several simplifying assumptions are taken to create these models. This paper evaluates different models for their suitability to model the InfiniBand™ network. All models are described and rated by comparing advantages and disadvantages for modelling InfiniBand™ . If no suitable model is found, the best currently known model is used as a base to derive a more accurate model.
Different models are described and evaluated with regard to small messages and the mentioned particularities in section 2.1, followed by the proposal of the new model and a parameter assessment benchmark in section 4. The results are summarized in section 5 and an outlook to future research is given.

1.1 Related Work

Many models have been developed in the past years. Most of them are dedicated to a specific hardware or network architecture [13, 2] or the shared memory paradigm (e.g. CICO [12, 7]). There are also some general purpose parallel models which try to stay architecture independent as the PRAM [5, 11], the BSP [17], the C^3 [9] or the LogP [3] model. These generic programming models are characterized and used as starting point for all further work. Several comparative studies and surveys are also available [16, 8, 1], but they provide only a limited view by comparing only a small subset of all available models.
Some groups are working on evaluating different models for different hardware architectures. For example Estefanel et. al. has shown in [4] that the LogP model is quite accurate for Fast Ethernet.

*thanks to Prof. Dr. Wolfgang Rehm for advising my work

But to the current knowledge of the author, there are no attempts to the generate a new more accurate model for InfiniBand™ .

2 Model Comparison

2.1 Organization

Each mentioned model is described by its main characteristics. A reference to the original publication is given if the reader is interested in further details (e.g. detailed information about execution time estimation). Each model is analyzed with regards to its advantages and disadvantages for modelling the InfiniBand™ architecture. A conclusion for further usage in the design process of a new model is drawn. Different enhancements by third authors have only a small impact on the accuracy of the model for small messages and are omitted. A full analysis can be found in the full version of the thesis [10]. The last section draws a conclusion and proposes a suitable model for small messages inside the InfiniBand™ network.

2.2 The PRAM Model

The PRAM model was proposed by Fortune et al. in 1978 [6]. It is the the most simple parallel programming model known. But there are some serious defects in its accuracy. It was mainly derived from the RAM model, which bases itself on the "Von Neumannn" model. It is characterized by P processors sharing a common global memory. Thus it can be seen as a MIMD[2] machine. It is assumed that all processors run synchronously (e.g. with a central clock) and that every processor can access an arbitrary memory location in one step. All costs for parallelisation are ignored, thus the model provides a benchmark for the ideal parallel complexity of an algorithm.

Evaluation:

The main advantage is the ease of applicability. But to reach this simplicity, several disadvantages have to be accepted. The main drawbacks are that all processors are assumed to work synchronously, that the interprocessor communication is nearly free[3] and that it neglects the contention when different cells in one memory module are accessed.
Thus, this model is not suitable for modelling InfiniBand™ because interprocessor communication is free.

2.3 The BSP Model

The Bulk Synchronous Parallel (BSP) model was proposed by Valiant in 1990 [17]. The BSP model divides the algorithm into several consecutive supersteps. Each superstep consists of a computation and a communication phase. All processors start synchronously at the beginning of each superstep. In the computation phase, the processor can only perform calculation on data inside its local memory[4]. The processor can exchange data with other nodes in the communication phase. Each processor may send at most h messages and receive at most h messages of a fixed size in each superstep. This is called a h-relation further on. A cost of $g*h$ (g is a bandwidth parameter) is charged for the communication.

Evaluation

Latency and (limited) bandwidth are modelled as well as asynchronous progress per processor. Each superstep must be long enough to send and receive the h messages[5]. This may lead to idle-time in some of the nodes if

[2] Multiple Instruction Multiple Data
[3] zero latency, infinite bandwidth leads to excessive fine-grained algorithms
[4] if this is data from remote nodes, it has been received in one of the previous supersteps
[5] the maximal h among all nodes!

the communication load is unbalanced. This contains the problem that messages received in a superstep cannot be used in the same superstep even if the latency is smaller than the remaining superstep length.
Because of the implicit synchronization, the BSP model is not suitable for modelling the usually asynchronous InfiniBand™ network.

2.4 The LogP Model

The LogP Model [3] was propsed by Culler et al. in 1993. It was developed in addition to the PRAM model (see chapter 2.2) to consider the changed conditions for parallel computing. It reflects different aspects of coarse grained machines which are seen as a collection of complete computers, each consisting of one or more processors, cache, main memory and a network interconnect[6]. It is based on four main parameters:

- L - communication delay (**upper** bound to the latency for NIC-to-NIC messages from one processor to another)
- o - communication overhead (time that a processor is engaged in transmission or reception of a single message, split up into o_s for send overhead and o_r for receive overhead)
- g - gap (indirect communication bandwidth, minimum interval between consecutive messages, $bandwidth \sim \frac{1}{g}$)
- P - number of processors

It is easy to understand that developing and programming in the PRAM model is easier than in the LogP model, but the bigger accuracy of this model should justify the additional effort.

Evaluation

The LogP model has several advantages over other models. It is designed for distributed memory processors and the fact that network speed[7] is far smaller than CPU speed. It is easily applicable for a flat network model[8]. It encourages careful scheduling of computation and overlapping communication as well as balanced network operations[9] which is very profitable for accuracy of determining the run time of many applications.
Some small drawbacks are that the whole communication model consists only of point-to-point messages. This does not respect the fact that some networks (especially InfiniBand™) are able to perform collective operations (e.g. multicast) ideally in $O(1)$.

2.5 Choosing a Model

As described in 2.4, the LogP model is the most accurate model in this specific case. Thus, it is used for all running time estimations in the following sections.
Several simplifying architectural assumptions can be made without lowering the asymptotical accurance of the model. Based on the fact that most clusters operate a central switch hierarchy which connects all nodes, the properties of this interconnect can be assumed as follows:

- full bisectional bandwidth
- full duplex operation (parallel send/receive)
- the forwarding rate is unlimited and packets are forwarded in a non-blocking manner
- the latency (L from LogP model) is constant above all messages
- the overhead (o) is constant for single messages (for simplicity: $o_s = o_r = o$)

[6] e.g. the Intel Delta or Paragon, Thinking Machines CM-5 ...
[7] this means latency as well as bandwidth
[8] central switch based, diameter = 1
[9] no single processor is ”flooded”

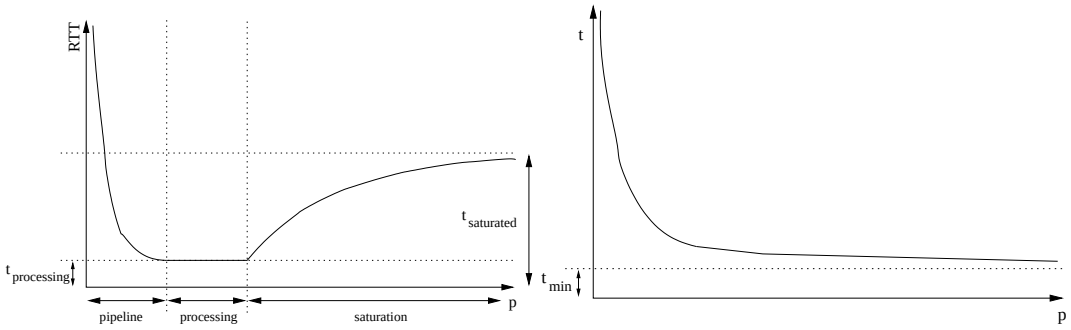

Figure 1: RTT (left) and Overhead (right) Model

3 Modelling the Architecture

A model of the RTT and overhead times for offloading based networks will be described to generalize the results of this work (e.g. to Myrinet, Elan) and to show the need for a new model. This model will be parametrized to fit to the test cluster mentioned in section 4.7.

A Model for the RTT

The RTT model consists of three sections, the warmup section for the NIC (e.g. pipelining or cache effects), the maximum performance section (NIC CPU is fully saturated) and the network saturation section. This model assumes that 1:n and n:1 communications are equal in terms of cost. The first section is typically represented by a pipeline startup function of the shape:

$$t_{pipeline} = \frac{\lambda_1}{\lambda_2 + p}$$

The second section is only defined by the maximum $CPU \rightarrow NIC \rightarrow NIC \rightarrow CPU$ throughput (packet processing rate), and is thus defined as constant:

$$t_{processing} = \lambda_3$$

The third section reflects the network saturation which typically behaves like an exponential function as:

$$t_{saturation} = \lambda_4 \cdot (1 - e^{\lambda_5 \cdot (p - \lambda_6)})$$

λ_4 and λ_5 influence the signature of the function and λ_6 introduces a p-offset.
Alltogether the RTT time can be described with the following abstract model, which is depicted in the left side of figure 1:

$$\begin{aligned} t_{rtt}(\lambda_{1\ldots6}) &= t_{saturation} + t_{processing} + t_{pipeline} \\ &= \frac{\lambda_1}{\lambda_2 + p} + \lambda_3 + \lambda_4 \cdot (1 - e^{\lambda_5 \cdot (p - \lambda_6)}) \end{aligned}$$

A Model for the Overhead

The send and receive overheads are modelled as pipeline startup functions. This is due to several cache effects and pipelining effects on the host CPU. The HCA should not be involved into this process, because the data is written into memory mapped registers inside the HCA memory.

The function can be described as:

$$t_{ov}(\lambda_{1...3}) \quad = \quad \lambda_1 + \frac{\lambda_2}{\lambda_3 + p}$$

and is depicted in the right part of figure 1.

Parametrizing the Model

The least squares method is used to find an optimal parametrization for all $\lambda_{1...6}$. This method calculates the sum of the squared deviations of the measured values to the functional prediction for all available data-points and tries to minimize it.
The approximation scheme for t_{ov} is omitted because it can be easily derived from the scheme shown above.

4 A new Model for InfiniBand™

As described in the analysis of the different models (see section 2.1), the LogP model reflects the needs to model an InfiniBand™ architecture quite well. The main assumptions, that each node consists of a complete "Von Neumann" computer with its processor, cache, memory and network interconnect and that the computing power is much higher than the network throughput are completely conformed by the InfiniBand™ architecture[10]. Unfortunately no presently known addition to the LogP model seems to be helpful for modelling small messages within InfiniBand™ networks, so the original LogP model has to be modified to fit our special needs.

4.1 Message Passing Options

For modelling all architectural details especially the different possibilities for sending and receiving data, the model has to be provided for each of the mentioned communication options separately.

4.2 The HCA Processor

The HCA[11] used to process previously posted work requests, participates actively in the communication and disburdens the host processor. which strictly lowers the o parameters. This implies an additional level of parallelism and introduces new possibilities for overlapping computation and communication. This is modelled as a part of the latency (L) parameter in the standard LogP model, which can be very accurate under most circumstances. But if the HCA is slower than the host CPU (the CPU is able post more work request than the HCA can process), contention will occur at the HCA[12] and the latency will vary from packet to packet (according to previously posted packets).

4.3 Hardware Parallelism

The InfiniBand™ standard proposes implicite hardware parallelism or pipelining to the vendors, therefore the easy idea of using a gap as time to wait between single messages cannot be very accurate in this architecture. The HCA can send two messages nearly in parallel until a single message fills up the whole bandwidth. Thus the linear model of LogP is not accurate enough. It is assumed that the gap is now part of the Latency which now depends on the number of previously issued send operations (denoted as L(p)). To reflect this behavior correctly, the model has to pay attention to the following send-receive scenarios:

- 1:1 communications
- 1:n communication

[10] at least with 4x links
[11] InfiniBand™ Host Channel Adapter
[12] the Queue Pairs in InfiniBand™ will fill up

- n:1 communication

The latter two can be implemented either by a consecutive post of single work requests or by a single post of a list of work requests. The performance implications have to be modelled also. The approximation function should behave like a normal pipeline startup functions with $t = a + \frac{b}{x}$. The parameters a and b have to be measured for each vendor specific InfiniBand™ solution.
The new model introduces the new parameter h for the time that the HCA spends to process a message (it can be subdivided into h_s and h_r for sender and receiver). The h parameters cannot be measured directly because all actions are performed in hardware without notificating the Host CPU. Thus, the model hides the h and g parameter inside the $L(p)$ parameter which varies now depending on the number of hosts addressed (p). This limits the model to be used only if one node does never send more than one packet to another node, because the $L(p)$ does only depend on the number of addressed hosts and not on the number of sent or received messages. This is for example given by the barrier problem and round-based algorithms. Further enhancements to the model to allow general use are part of the future work.
The traditional LogP would be a linear function like: $L(p) = h_s(p) + L + (p-1) * g + h_r(p)$, but because of the parallelism and pipeline effects, this function is assumed to be non-linear. This parameter is measureable and can be used to find the best algorithm for problems based on small messages with InfiniBand™ . The new model, named LoP, is depicted in figure 2. It has to be mentioned that $L(p)$ and o_s/o_r overlay each other because they are processed on different CPUs. Thus the HCA starts immediately to process messages after the CPU posted the first one. It is assumed that $o \ll L(p)\ \forall p \in \mathbb{N}$. The only exception is the VAPI call to post a list of requests, where the HCA has to wait until all requests have been posted, because all are posted at once.

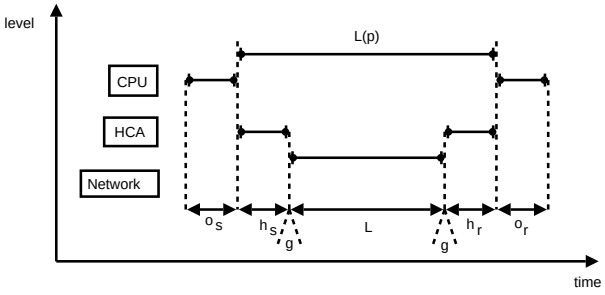

Figure 2: A new Model for InfiniBand™

4.4 Measuring the Parameters

All parameters mentioned in the previous section are hardware specific and have to be measured for each machine. The only possibility to measure this parameters is to evaluate the running time of different operations performed by the HCA. The following statements are based on a typical interaction with the HCA. The parameters can be measured as follows:

- $o_s(p)$ - time to complete the call `VAPI_post_sr()` or `EVAPI_post_sr_list()`
- $o_r(p)$ - time to complete the call `VAPI_post_rr()` or `EVAPI_post_rr_list()`
- $L(p) = \frac{RTT(p)}{2} - (o_s(p) + o_s(1))$ (sending to p processors and receiving from p processors - the HCA starts processing after the first request arrived - the exception for posting a list of send requests is found below, $o_r(p)$ does not matter because Receive Requests can be posted in advance)

- $L^{list}(p) = \frac{RTT(p)}{2} - (p \cdot o_s(p) + o_s(1))$ (sending to p processors and receiving from p processors for posting a list of send requests)

4.5 Benchmarking the Parameters

The only way to verify the model and to measure the parameters defined in the LoP model is to benchmark the actual hardware. The used benchmark, written in C with MPI support is presented in the following section.

4.6 Benchmark Structure

The benchmark implements the scheme described in section 4.4. It uses two different scenarios to measure all necessary parameters. Scenario 1 is used to measure all overheads for sending a single message, while scenario 2 measures ping-pong times for 1:n and n:1 communications. The exact time is measured by using the RDTSC CPU instruction which counts the cycles of the CPU. This makes the benchmark not portable to architectures which are not i386 compatible.

4.7 Benchmark Results

All benchmarks are extremly implementation specific. The measured values highly depend on the given architecture and circumstances. All following benchmark results have been metered on a 64 node InfiniBand™ cluster, interconnected with a 64 port switch (Mellanox MTS 9600 switch, 3GHz Xeon nodes with MTPB 23108 InfiniBand™ adapters). The general architecture to assess the parameters L and o of the LoP model for offloading based systems is modelled in the following section.
The benchmarks have been conducted for Send/Receive and RDMA Write without immediate operation. RDMA Read and RDMA Write with immediate have not been considered because the architectural design and several studies ([15], [14]) show that these operations are generally slower than RDMA Write without immediate. Atomic Operations are not available on the used HCAs.

Send/Receive Results

The minimal RTT results of Send/Receive InfiniBand™ operations can be seen in figure 3. The depicted func-

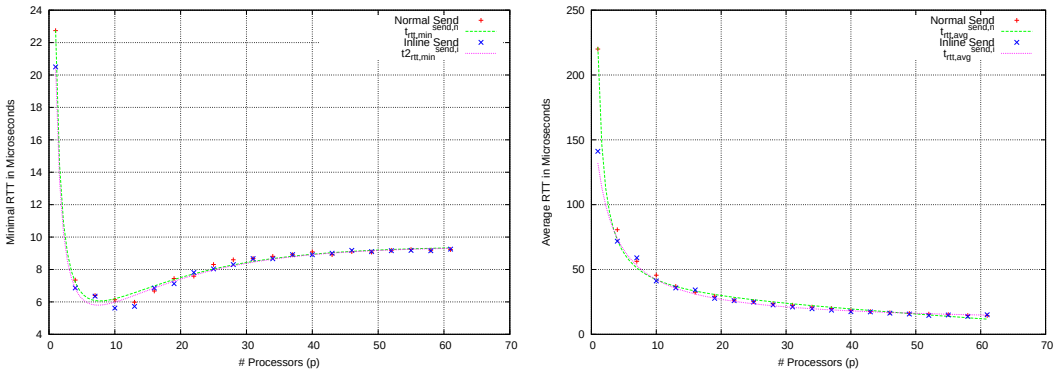

Figure 3: Minimal and Average Send/Receive RTT Times

tion describes t_{rtt} as described in section 3. The measured and fitted functions are shown in figure 3 and

mathematically described in the following:

$$t_{rtt,min}^{send,n}(p) \quad = \quad 9.1637 + \frac{22.4558}{-0.0140 + p} + 0.0174 \cdot \left(1 - e^{-0.0625 \cdot (p - 101.3065)}\right)$$

The difference between normal and inline send is modelled quite accurate. It is constantly about $1\mu s$ for small processor counts and vanishes when the network begins to saturate ($p \approx 30$).
The measured send (o_s) and receive (o_r) overheads are omitted, because they equal to the RDMA results shown in Figure 5. The fastest method to post more than two send requests is generally to post a list of send requests. All other methods could be beneficial with special send operations (inline send).

RDMA Write Results

The minimal and average RTT results of RDMA Write InfiniBand™ operations can be seen in figure 4. The

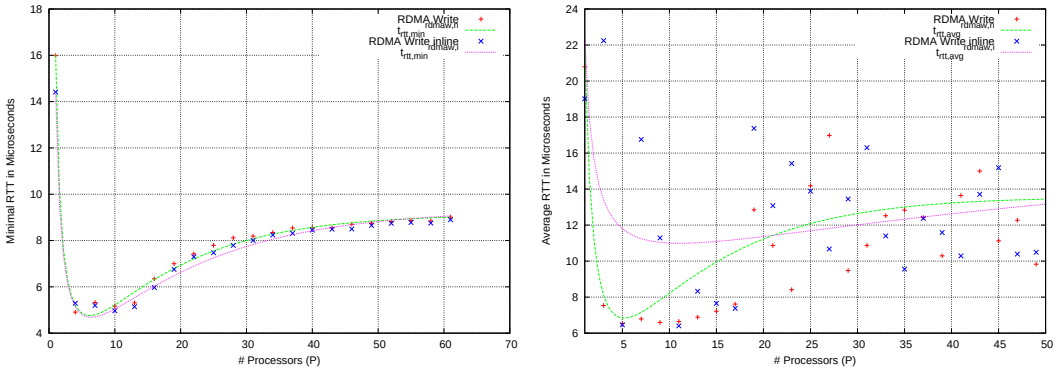

Figure 4: Minimal and Average RDMA Write RTT Times

shown function depicts t_{rtt} as described in section 3 for RDMA Write operation. The average functions show a big deviation, and are only plotted and fitted up to 50 processors. The t_{rtt} values raise quickly up to $700\mu s$ for bigger processor counts, which could lead to the conclusion that harsh memory or bus contention occurs. The plotted deviation may be caused by memory contention and blocking/arbiting effects of single RDMA write operations and varies extremely between different measurements.
The inline send is again about $1\mu s$ faster than the normal send for small processor counts p and this difference vanishes during the network saturation ($p > 30$). The normal send seems to be much better and even more ”stable” in the average case than the inline send. The functions for the average case are also quite accurate, even if the measured values oscillate a lot. This is guaranteed by the least squares method, which punishes bigger deviations more than smaller ones.
The fitted functions for all described data-sets are given in the following:

$$t_{rtt,min}^{rdmaw,n}(p) \quad = \quad 4.4642 + \frac{16.7937}{0.0058 + p} + 4.4751 \cdot \left(1 - e^{-0.0642 \cdot (p - 12.9209)}\right)$$

Figure 5 shows the send overhead (o_s) for RDMA Write operations. Posting a list of send requests is again the fastest method of sending multiple packets, but to send the data inline could lower the latency in the best case.

$$t_{srov}^{rdmaw,n}(p) \quad = \quad 0.5557 + \frac{0.2103}{-0.7728 + p}$$

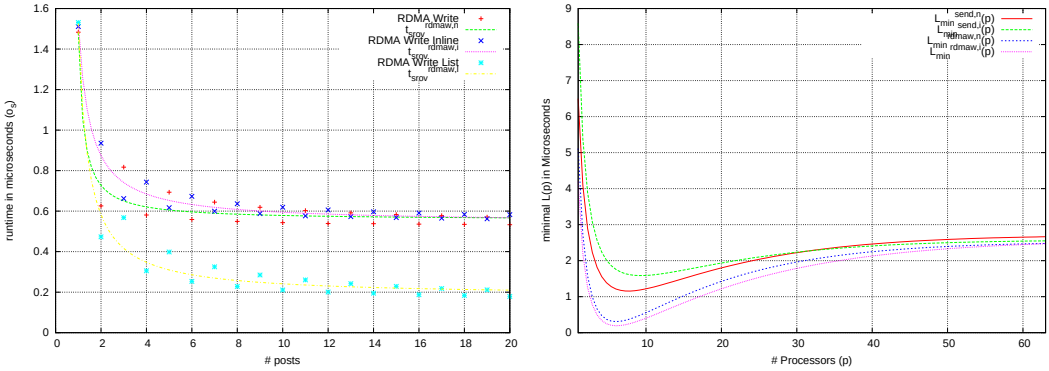

Figure 5: RDMA o_s overhead and minimal $L(p)$ overhead

Calculating the L(p) Parameter

The $L(p)$ parameter is calculated as described in section 4.4 and results in the equations shown in the following (only $L_{min}^{send,n}(p)$ and due to space limitations)

$$\begin{aligned} L_{min}^{send,n}(p) &= \frac{t_{rtt,min}^{send,n}(p)}{2} - \left(t_{sr,ov}^{send,n}(1)\right) - \left(t_{sr,ov}^{send,n}(p)\right) \\ &= 4.58 + \frac{11.23}{-0.01+p} + 0.01 \cdot \left(1 - e^{-0.06 \cdot (p-101.31)}\right) - \left(0.52 + \frac{0.84}{-0.12+1}\right) - \left(0.52 + \frac{0.84}{-0.12+p}\right) \end{aligned}$$

All functions for the different possibilities to send or receive 1 byte packets using the send-receive semantics are shown in figure 5.
Thus the time to send 1 message to n hosts for each possible post send request / send type combination can be assessed with:

$$t_{1:n} = o(n) + n \cdot L(n)$$

. . . for posting a list of send requests:

$$t_{1:n}^{list} = n \cdot o(n) + n \cdot L(n)$$

5 Conclusions and Future Work

This work shows the analysis of small message performance of InfiniBand™ and the development of a new and very accurate model. It shows that the LogP model is quite accurate for a big number of nodes. But the LoP model offers different optimization chances, for example by showing the advantages of sending more than one message per round. This new model made it possible to enhance the performance of the barrier operation for InfiniBand™ up to 40% in comparison to the best known solution [10].
The next steps include the evaluation of the LogP model (and its modifications) for a variable message size and the simplification of the very complicated equations of the LoP model to enhance its ease of use.
The full InfiniBand™ model including all derived equations and the process of modelling different barrier algorithms and developing a new, more efficient algorithm can be found in the original thesis [10].

References

[1] G. Bilardi, K. T. Herley, and A. Pietracaprina. BSP vs LogP. In *SPAA '96: Proceedings of the eighth annual ACM symposium on Parallel algorithms and architectures*, pages 25–32. ACM Press, 1996.

[2] G. Blelloch. Scans as Primitive Operations. In *Proc. of the International Conference on Parallel Processing*, pages 355–362, August 1987.

[3] D. Culler, R. Karp, D. Patterson, A. Sahay, K. E. Schauser, E. Santos, R. Subramonian, and T. von Eicken. LogP: towards a realistic model of parallel computation. In *Principles Practice of Parallel Programming*, pages 1–12, 1993.

[4] L. A. Estefanel and G. Mounie. Fast Tuning of Intra-Cluster Collective Communications. In *Recent Advances in Parallel Virtual Machine and Message Passing Interface: 11th European PVM/MPI Users Group Meeting Budapest, Hungary, September 19 - 22, 2004. Proceedings*, 2004.

[5] S. Fortune and J. Wyllie. Parallelism in random access machines. In *STOC '78: Proceedings of the tenth annual ACM symposium on Theory of computing*, pages 114–118. ACM Press, 1978.

[6] S. Fortune and J. Wyllie. Parallelism in random access machines. In *STOC '78: Proceedings of the tenth annual ACM symposium on Theory of computing*, pages 114–118. ACM Press, 1978.

[7] P. G. Gibbons, Y. Matias, and V. Ramachandran. Can a shared memory model serve as a bridging model for parallel computation? In *ACM Symposium on Parallel Algorithms and Architectures*, pages 72–83, 1997.

[8] S. E. Hambrusch. Models for parallel computation. In *ICPP Workshop*, pages 92–95, 1996.

[9] S. E. Hambrusch and A. A. Khokhar. An architecture-independent model for coarse grained parallel machines. In *Proceedings of the 6-th IEEE Symposium on Parallel and Distributed Processing*, 1994.

[10] T. Hoefler. Evaluation of publicly available Barrier-Algorithms and Improvement of the Barrier-Operation for large-scale Cluster-Systems with special Attention on InfiniBand^{TM}Networks. Master's thesis, TU-Chemnitz, 2004. http://archiv.tu-chemnitz.de.

[11] R. M. Karp and V. Ramachandran. Parallel algorithms for shared-memory machines. In J. Leeuwen, editor, *Handbook of Theoretical Computer Science: Volume A: Algorithms and Complexity*, pages 869–941. Elsevier, Amsterdam, 1990.

[12] J. R. Larus, S. Chandra, and D. A. Wood. CICO: A Practical Shared-Memory Programming Performance Model. In Ferrante and Hey, editors, *Workshop on Portability and Performance for Parallel Processing*, Southampton University, England, July 13 – 15, 1993. John Wiley & Sons.

[13] F. T. Leighton. *Introduction to parallel algorithms and architectures: array, trees, hypercubes*. Morgan Kaufmann Publishers Inc., 1992.

[14] J. Liu, W. Jiang, P. Wyckoff, D. K. Panda, D. Ashton, D. Buntinas, W. Gropp, and B. Toonen. Design and Implementation of MPICH2 over InfiniBand with RDMA Support. In *Int'l Parallel and Distributed Processing Symposium, Proceedings*, 2004.

[15] J. Liu, J. Wu, and D. K. Panda. High Performance RDMA-Based MPI Implementation over InfiniBand. *Int'l Journal of Parallel Programming, 2004*, 2004.

[16] B. M. Maggs, L. R. Matheson, and R. E. Tarjan. Models of Parallel Computation: A Survey and Synthesis. In *Proceedings of the 28th Hawaii International Conference on System Sciences (HICSS)*, volume 2, pages 61–70, 1995.

[17] L. G. Valiant. A bridging model for parallel computation. *Commun. ACM*, 33(8):103–111, 1990.