# Toward Performance Models of MPI Implementations for Understanding Application Scaling Issues

Torsten Hoefler[1], William Gropp[1], Rajeev Thakur[2], and Jesper Larsson Träff[3]

[1] University of Illinois at Urbana-Champaign, IL, USA,
{htor,wgropp}@illinois.edu
[2] Argonne National Laboratory, Argonne, IL, USA
thakur@mcs.anl.gov
[3] Dept. of Scientific Computing, University of Vienna, Austria
traff@par.univie.ac.at

**Abstract.** Designing and tuning parallel applications with MPI, particularly at large scale, requires understanding the performance implications of different choices of algorithms and implementation options. Which algorithm is better depends in part on the performance of the different possible communication approaches, which in turn can depend on both the system hardware and the MPI implementation. In the absence of detailed performance models for different MPI implementations, application developers often must select methods and tune codes without the means to realistically estimate the achievable performance and rationally defend their choices. In this paper, we advocate the construction of more useful performance models that take into account limitations on network-injection rates and effective bisection bandwidth. Since collective communication plays a crucial role in enabling scalability, we also provide analytical models for scalability of collective communication algorithms, such as broadcast, allreduce, and all-to-all. We apply these models to an IBM Blue Gene/P system and compare the analytical performance estimates with experimentally measured values.

## 1 Motivation

Performance modeling of parallel applications leads to an understanding of their running time on parallel systems. To develop a model for an existing application or algorithm, one typically constructs a dependency graph of computations and communications from the start of the algorithm (input) to the end of the algorithm (output). This *application model* can then be matched to a *machine model* in order to estimate the run time of the algorithm on a particular architecture.

Performance models can be used to make important early decisions about algorithmic choices. For example, to compute a three-dimensional Fast Fourier Transformation (3d FFT), one can either use a one-dimensional decomposition where each process computes full planes (2d FFTs) or a two-dimensional decomposition where each process computes sets of pencils (1d FFTs). If we assume

an $N^3$ box and $P$ processes, the 1d decomposition only requires a single parallel transpose (alltoall) and enables the use of more efficient 2d FFT library calls (for example, FFTW) but is limited in scalability to $P \leq N$. A 2d decomposition scales up to $P \leq N^2$ processes but requires higher implementation effort and two parallel transpose operations going on in parallel on subsets of the processes [6]. Thus, the best choice of distribution mostly depends on the communication parameters and the number of processes.

Application designers typically have a basic understanding and expectation of the performance of some operations on which they base their algorithmic decisions. For example, most application developers would assume that transmitting a message of size $S$ has linear costs in $S$ and that broadcasting a small message to $P$ processes would take logarithmic time in $P$. Such simple assumptions of *performance models*, or general *folklore*, often guide application models and algorithm development.

One problem with this approach is that inaccuracies in the model often do not influence the medium-scale runs in which they are verified but have significant impact as the parameters ($S$ or $P$) grow large. One particular example is that several application models assume that broadcast or allreduce scale with $\Theta(S \log(P))$ (e.g., [3, 17]) while, as demonstrated in Section 4, a good MPI implementation would implement a broadcast or allreduce with $\Theta(S + \log(P))$ [5, 13, 21]. Generally speaking, performance models for middleware libraries such as MPI depend on the parameters of the network (e.g., bandwidth, latency, topology, routing) **and** the implemented algorithms (e.g., collective algorithms, eager and rendezvous protocols) and are thus hard to generalize.

In this paper, we advocate the importance of communication models for MPI implementations at large scale. We contend that such models should be supplied by each MPI implementation to allow users to reason about the performance. We sketch a hierarchical method to derive communication models of different accuracies so that an implementer can trade off the effort to derive such a model with the accuracy of the model. We provide guidance for MPI modeling by demonstrating that simple asymptotic models can be very helpful in understanding the communication complexity of a parallel application. We also mention some pitfalls in modeling MPI implementations.

## 2 Previous Work and a General Approach to Modeling

Designing a performance model for MPI communication is a complex task. Numerous works exist that model the performance of a particular MPI implementation on a particular system [1, 15, 16, 19, 20]. Other works focus on modeling particular aspects of MPI, such as collective communication [11, 15, 18].

Those works show that accurate models are only possible in very limited cases and require high effort. Handling the whole spectrum, even for a particular MPI implementation on one particular system, is very challenging. In the extreme case, such a model would require the user to consider all parameters of a parallel application run that are (intentionally) outside the scope of MPI (e.g., process-to-node mappings or contention caused by traffic from other jobs).

However, many of those parameters may have little influence on a useful model and might thus be abstracted out. In this work, we build upon common modeling methodologies from previous work and design a hierarchy of models that allows one to trade-off design effort for accuracy. Our work is intended to encourage and guide MPI implementers to specify performance characteristics of each implementation. Below we present different approaches at different levels of detail.

**Asymptotic Model** A useful first approximation of communication performance is to give the asymptotic time complexity of the communication operations. We use the standard $O$, $\Theta$, and $o$ notation for asymptotic models. Asymptotic models can often be deduced with relativey little effort and allow assumptions about the general scalability, but do not allow for absolute statements because the constants remain unspecified. For example, an MPI implementation could state that the implemented broadcast scales with $T_{BC}(S,P) = \Theta(S + \log(P))$.

**Dominant term exact model** Sometimes it might be possible (or desirable) to indicate that the significant terms are known, while lower-order terms are either not known or do not play a role asymptotically. An optimal broadcast algorithm that fully exploits the bandwidth of the underlying (strong) interconnect could thus be specified as having running time $T_{BC}(S,P) = \Theta(\log(P)) + \beta S$ [13] in contrast to merely efficient broadcast algorithms with the same asymptotic performance $T_{BC}(S,P) = \Theta(S+\log(P))$, but in reality behaving as $T_{BC}(S,P) = \Theta(\log(P)) + 2\beta S$ [5].

**Bounded (Parametrized) Model** It is often possible to specify some of the constants for the asymptotic model and thus allow absolute statements. Such constants often depend on many parameters, and it might be infeasible to specify all of them (e.g., process-to-node mapping or contention). Thus, we propose to specify parameterized upper and lower bounds (e.g., for the worst-possible and best-possible mappings) for those costs. Such models allow the user to make absolute statements about parallel algorithms under worst and best conditions. For example, Equation (2) gives such an estimate for point-to-point messages under congestion.

**Exact (Parametrized) Model** It might be possible to define exact models for operations in some specified settings (for example, dedicated network links). This is the most accurate technique in our hierarchy and most convenient for the application developer. For example, the cost of a barrier on a BlueGene/P (BG/P) is $T_{BAR} = 0.95\mu s$ [7], independent of $P$.

*Parameter Ranges* Implementations might adapt the communication algorithms based on the input parameters. For example, point-to-point communications are often implemented with two protocols, eager and rendezvous, depending on the

size of the message. Implementers can simply specify different models for different parameter regions as shown in Equation (1).

In the remainder of the paper, we discuss a modeling strategy for point-to-point operations (Section 3) and collective operations (Section 4). We show common pitfalls that are often ignored and demonstrate their influence in practice. We focus on developing guidelines for modeling, similar to the guidelines for performance measurement in [8]. To illustrate the techniques and to demonstrate the importance of certain aspects of modeling, we use a BG/P system as an example where we find it helpful. However, we do not specify a complete communication model for BG/P.

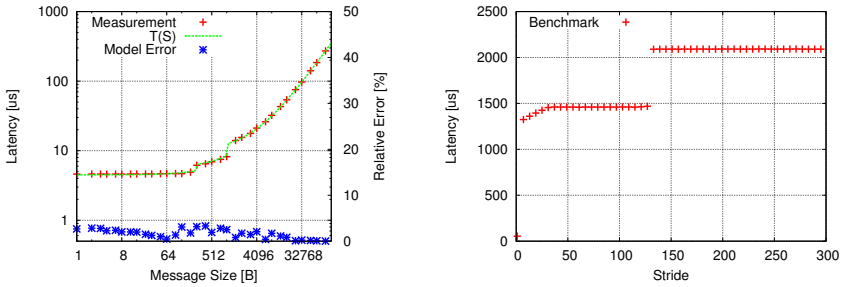
## 3 The Deficiency of Current Point-to-Point Models

The asymptotic model for point-to-point communication is typically $T(S) = \Theta(S)$. A good parametrized model could be the LogGP model [2]. The simpler latency-bandwidth model $T = \alpha + S\beta$ is covered by setting $g = 0$, $\beta = G$ to the time to transmit a single byte, and $\alpha = L + 2o$ to the start-up overhead. Those models are well understood and thus not further discussed. Figure 1(a) shows an example for an accurate congestion-free point-to-point communication on BG/P. The simple point-to-point model would have three components (we use a piecewise linear latency-bandwidth model):

$$T(S) = \begin{cases} 4.5\mu s + 2.67ns/B \cdot S : & S \leq 256B \\ 5.7\mu s + 2.67ns/B \cdot S : & 256B < S \leq 1024B \\ 9.8\mu s + 2.67ns/B \cdot S : & 1024B < S \end{cases} \quad (1)$$

However, for modeling real applications such models suffer from the following shortcomings:

*Overlap and Progress* A point-to-point model should cover the ability of the MPI library to overlap computation and communication. The LogGP model provides the parameter $o$ to model the per-message overhead; however, this might not be sufficient if the overhead grows with the message size. An additional parameter, for example, $O$ for the overhead per byte, could be introduced to capture those effects [15]. Such a model and its derivation are well understood, and we omit details for brevity.



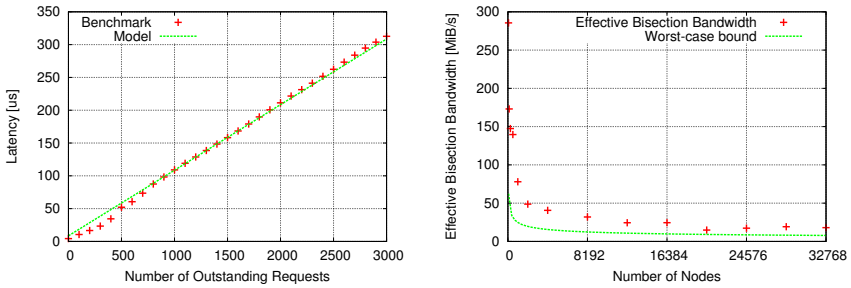

(a) Latency Model.

(b) 16 kiB with different strides.

**Fig. 1.** Example Models for Latency and Datatypes on BlueGene/P.

*Synchronization* For some applications, it is important to know about the synchronization properties of messages. MPI implementations typically use receiver buffering for small messages and, for large messages, a rendezvous protocol that delays the sender until the receiver is ready. This effect can be modeled easily and is covered by the parameter $S$ in the LogGPS model [12].

*Datatypes* MPI offers a rich set of primitives to define derived datatypes for sending and receiving messages. As these datatypes can reach high complexities, the time to gather all the data from memory can vary significantly for different datatypes. Figure 1(b) shows the influence of the stride in a simple vector datatype when sending 16 kiB MPI_CHAR data. The contiguous case takes $53\mu s$, while a stride of 1 increases the latency to $1292\mu s$ due to the $\approx 2^{14}$ memory accesses. The memory hierarchy in modern computer systems makes modeling datatypes complex; however, one can often provide a worst case that is bound by the slowest memory. In our example, the worst-case ($2090\mu s$) is reached at a stride of 128 when the buffer exceeds the L3 cache on BG/P.




(a) Matching Queue. (b) Congestion.

**Fig. 2.** Example Models for Matching and Congestion on BlueGene/P.

*Matching Queue Length* The length of the matching queue can dramatically influence performance as shown in Figure 2(a). We represent the worst-case overhead of a matching queue with $R$ outstanding messages on BG/P as $T_{match}(R) \leq 100ns \cdot R$. A simple 27-point stencil would cause a traversal of an average of 13 requests, adding $T_{match}(13) = 1.3\mu s$ to the latency before the right message is matched.

*Topology (Mapping)* The network topology and the process-to-node mapping is also of crucial importance [4]. Information about topology and potentially the mapping is needed for an accurate point-to-point model. For a torus, for example, one could simply multiply the latency with the number of hops. We note that $L$ in the LogP model, by definition, represents the upper bound (i.e., the maximum latency between any two endpoints).

*Congestion* Point-to-point models often ignore network congestion, which might be problematic for certain communication patterns. Even networks with full bisection bandwidth are not free of congestion [10]. One way to model congestion would be to define the *effective bisection bandwidth* (formally defined in [10]) as upper bound for $1/G$.

It is now clear that Equation (1) represents the ideal case: free of congestion, minimal queue lengths, consecutive memory accesses, and an optimal mapping. We will now present an example model for congestion in the general case. First, since we know the upper bound to the bandwidth (Equation (1)), we derive the lower bound, that is, the maximum congestion possible. For this purpose, we assume a cubic allocation on a 3-d torus ($k$-ary 3-cube) network of size $N_x = N_y = N_z = k$, and (ideal) adaptive routing along shortest paths. Per convention, nodes are identified by three digits ranging from $0 \dots k-1$. In order to cause the maximum congestion in the network, we choose pairs with maximum distance. In a $k$-ary 3-cube, the maximum distance between two nodes is $d_3 = 3 \cdot \lfloor k/2 \rfloor$. We assume that each node can inject into all of its six links simultaneously and present the following simple argument: Each node injects six packet streams along the shortest paths to a node at distance $d_3$ (such a pattern can be generated by connecting each node at coordinate $xyz$ with another node at distance $\lfloor k/2 \rfloor$ in each dimension). Since each stream occupies $d_3$ links, a total of $6 \cdot k^3 \cdot d_3 = 9k^4$ links would be needed for a congestion-free routing (WLOG, we assume that $k$ is even). Now, we assume that the traffic is spread evenly across the $6 \cdot k^3$ total (unidirectional) links in the steady state of our ideal adaptive routing. This results in a congestion of $3/2k = \mathcal{O}(\sqrt[3]{P})$ per link. Thus, the model presented in Equation (1) must be corrected to reflect congestion on $P$ processes, for example for $S > 1024B$:

$$9.8\mu s + 2.67^{ns}/_{B} \cdot S \leq T(S,P) < 9.8\mu s + 2.67^{ns}/_{B} \cdot S \cdot 3/2 \sqrt[3]{P} \qquad (2)$$

Upper and lower bounds for the other characteristics can be derived similarly. It is sometimes important to determine the average case congestion/bandwidth, for example, to estimate the scalability of pseudo-random communication patterns as found in many parallel graph computations. The *effective bisection bandwidth* [10] is a good average-case metric and can be measured by benchmarking a huge number of bisection communications between random subsets of processes. Figure 2(b) shows the effective bisection bandwidth as benchmarked on BG/P. The figure also shows the bandwidth bound for the worst-case mapping ($2/3 \sqrt[-3]{P} \cdot 374.5^{MiB}/_{s}$) assuming ideal routing.

## 4 Performance Models for Collective Communication

Models for collective communication are of crucial importance for analyzing the scalability of parallel applications. Collective models are often simpler to use than accurate point-to-point models because the algorithms are fixed. Thus, parameters such as topology, synchronization, congestion, and the matching queue

can usually be hidden from the user. Asymptotic bounds can often be derived easily from the implementation. A "high quality" MPI implementation should make such statements.

More accurate parametrized models can be specified with a simple extension of point-to-point models as proposed by Xu [22]. Here, one would simply model all parameters, such as $L$ and $G$ in the simplified model, as dependent on the number of processes in the collective and the operation type. An all-to-all communication could be expressed as $T_{a2a} = \alpha(P) + S \cdot \beta(P)$. In a network with full effective bisection bandwidth, one could set $\beta(P) = G$. Startup overheads in $\alpha$ would likely scale linearly in $P$ ($\alpha = \Omega(P)$) for larger messages. However, such models that allow arbitrary functions as parameters in the general case (even tables) are often hard to use to analyze scaling in practice.

Another good method for modeling collective algorithms that build upon point-to-point methods is to construct the model from point-to-point models [9, 11,18]. Note that we do not prescribe a specific model rather than a methodology to design such models. Some special networks or topologies might require the addition of terms to describe certain effects (e.g., contention in a torus network). If significant effects are too complex to describe in such a model (e.g., process-to-node mappings) then the upper and lower bounds (best and worst case) should be given.

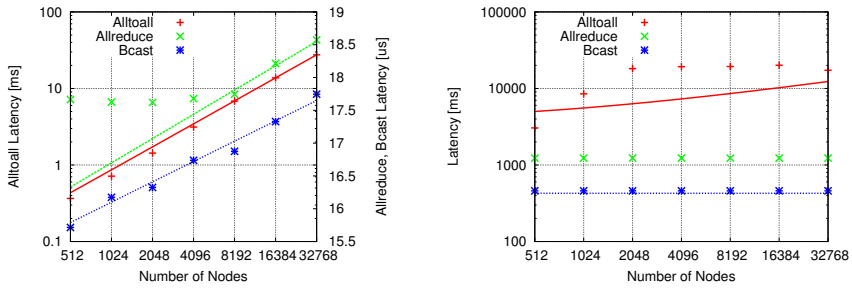
*Process-to-Node Mapping* The process-to-node mapping often plays a role in the performance of collective communication. For instance, the performance of rooted collectives, such as broadcast, can depend on the position of the root and the network topology. This is especially important for multi-core systems. Also, on BG/P the performance varies by the type of allocation. For example, for broadcast, performance was degraded to half for non-cubic allocations.

*Datatypes* A similar discussion as for point-to-point models applies.

Let us discuss three example models for MPI_Bcast, MPI_Allreduce, and MPI_Alltoall on BG/P. We do this by comparing theoretical bounds based on first principles gathered from the documentation [7] with benchmarked performance on MPI_COMM_WORLD on full allocations. Models for collective communications on other communicators and non-cubic allocations could be derived with a similar method. All benchmarks used the synchronous BG/P hardware barrier to start the operation on all processes, measured the time for a single execution, and report the average time.

**Small Data** First, we look at operations with a single integer (8 bytes), where the specialized collective network is used. The bandwidth of the collective network is 824 MiB/s [7]. IBM, however, did not release latency numbers for this network; thus, we resort to numerical methods for deriving a model.

For the **broadcast** time, we assume $T_{BC}(P, 8) = \alpha_T + \beta_T^{BC} \log_2(P)$ for the collective tree network. $\alpha_T$ models the startup overhead and $\beta_T$ models the cost (latency) per stage of the tree. With a small broadcast on two processes, we determined $\alpha_T = 13\mu s$ and $\beta_T^{BC} = 0.31\mu s$ from a large run with $P = 32,000$.




(a) 8 Byte per process. (b) Constant 960 MiB total.

**Fig. 3.** Small data (8 Byte process) and large data (960 MiB) scaling for different collective operations.

For **allreduce**, we used a similar model and determined $\beta_T^{SUM} = 0.37\mu s$ (the difference of $60ns$ is most likely caused by the higher overhead of the Integer sum): $T_{ARE}(P, 8) = \alpha_T^{SUM} + \beta_T^{SUM} \log_2(P)$. We found that for $P \leq 4,096$, the time is a constant $17.77\mu s$, which seems to indicate some other constant overhead in the implementation that probably overlaps with the communication in the tree network.

For **alltoall**, the implementation simply sends to all peer processes [7]; thus, we assume $T_{A2A}(P, 8) = \alpha + g(P - 1)$. As noted in Section 3, $\alpha = L + 2o$, and we determined $g = 0.84\mu s$ for $P = 32,000$. The model functions and the measurement results are shown in Figure 3(a) as points and lines, respectively.

**Large Data** For the large-data collectives, we communicated the maximum possible buffer size of $960MiB$ on BG/P (for alltoall, $960MiB/P$ per process).

Large-data **broadcasts** use all six links of the Torus network and the deposit-bit feature for communication. The maximum effective bandwidth would be $6 \cdot 374.5MiB/s = 2247MiB/s$ and is shown as a line in Figure 3(b). The broadcast would need at least $T_{BC}(P, 8)$ to reach all endpoints. We thus extend our broadcast model with a bandwidth term: $T_{BC}(P, S) = \alpha_T + \beta_T^{BC} \log_2(P) + \frac{2.67}{6} ns/B \cdot S$.

Large **allreduces** use the same message pattern as broadcast, but each stage is slower because of the reduction operation. An experiment showed that the allreduces takes $T_{ARE}(P, 9.6 \cdot 10^8) = 2.68 \cdot T_{BC}(P, 9.6 \cdot 10^8)$.

Let us now recapitulate an argument for the best-case **alltoall** bandwidth assuming ideal adaptive routing. As stated before, alltoall is implemented by simply sending from all processes to all other processes (in some order). We thus assume that all messages hit the network at the same time and are either limited by the injection at the endpoints or by the congestion in the network. Lam et al. showed bounds for one- and two-dimensional tori in [14]. We model the time with LogGP: $T_{A2A} = (P-1)g + SG \cdot \max\{\frac{P-1}{6}, C(P)\}$ and a congestion factor $C(P)$ (we assume $g > o$ WLOG). For alltoall, we assume that all messages contribute to the worst-case congestion in the network. On a $k^3$ grid (for odd $k$, WLOG), those messages occupy different numbers of links, depending on their Euclidean distance, with a maximum of $d_3 = \frac{3(k-1)}{2}$. Let $d = d_1 = \frac{k-1}{2}$, then the total

number of occupied links is

$$N(k) \leq k^3 \cdot 2 \sum_{x=0}^{d} 2 \sum_{y=0}^{d} 2 \sum_{z=0}^{d} (x+y+z) = k^3 12d\,(d+1)^3 = \mathcal{O}(k^7).$$

With a total of $6k^3$ links in the torus, the congestion per link (assuming ideal routing) $C(k) \leq N(k)/6k^3 = 2d(d+1)^3 = \mathcal{O}(k^4)$ and with $k^3 = P$, $C(P) \leq \sqrt[3]{P}(\sqrt[3]{P}/2+1)^3 = \mathcal{O}(P\sqrt[3]{P})$. Thus the lower bound for a bandwidth-bound alltoall on a torus would be:

$$T_{A2A} \geq g(P-1) + SG\sqrt[3]{P}(\sqrt[3]{P}/2+1)^3.$$

Figure 3(b) shows the bound and benchmark results for alltoall. This analysis indicates that potential for further optimization exists in BG/P's alltoall implementation. Faraj et al. suggest that increasing the number of FIFOs would mitigate the problem [7].

## 5 Summary and Conclusions

In this work, we describe the importance of analytic performance models for MPI implementations. Such models and their accuracy become more important in the context of algorithm and application design and validation on very large (petascale or exascale) systems. We argue that MPI implementers should supply analytic models with an MPI library in order to allow users to make algorithmic decisions and analyze scalability.

We described a hierarchy of modeling approaches that allow the designer to trade accuracy against effort, and we argue that asymptotic models would already provide important hints to application developers. We demonstrate how simple performance models can be developed, discuss common pitfalls, and show how to address those issues with examples on the BG/P architecture.

Performance is the main motivator for parallelization, and thus, performance modeling is most important in the context of MPI. Our work motivates a discussion of performance models in the MPI community and provides some initial guidance towards more useful modeling for MPI.

**Acknowledgments** This work was supported in part by the Office of Advanced Scientific Computing Research, Office of Science, U.S. Department of Energy, under contract DE-AC02-06CH11357 and DE-FG02-08ER25835, and by the Blue Waters sustained-petascale computing project, which is supported by the National Science Foundation (award number OCI 07-25070) and the state of Illinois.

## References

1. Al-Tawil, K., Moritz, C.A.: Performance modeling and evaluation of MPI. Journal of Parallel and Distributed Computing 61(2), 202–223 (2001)
2. Alexandrov, A., Ionescu, M.F., Schauser, K.E., Scheiman, C.J.: LogGP: Incorporating long messages into the LogP model for parallel computation. Journal of Parallel and Distributed Computing 44(1), 71–79 (1997)

3. Barker, K.J., Davis, K., Kerbyson, D.J.: Performance modeling in action: Performance prediction of a Cray XT4 system during upgrade. In: Proceedings of the 2009 IEEE Intl. Symp. on Parallel&Distributed Processing. pp. 1–8 (2009)
4. Bhatelé, A., Bohm, E., Kalé, L.V.: Topology aware task mapping techniques: An API and case study. SIGPLAN Not. 44(4), 301–302 (2009)
5. Chan, E., Heimlich, M., Purkayastha, A., van de Geijn, R.A.: Collective communication: theory, practice, and experience. Conc. & Comp. 19(13), 1749–1783 (2007)
6. Eleftheriou, M., Fitch, B.G., Rayshubskiy, A., Ward, T.J.C., Germain, R.S.: Scalable framework for 3D FFTs on the Blue Gene/L supercomputer: implementation and early performance measurements. IBM J. Res. Dev. 49(2), 457–464 (2005)
7. Faraj, A., Kumar, S., Smith, B., Mamidala, A., Gunnels, J.: MPI collective communications on the Blue Gene/P supercomputer: Algorithms and optimizations. In: 17th IEEE Symposium on High-Performance Interconnects. pp. 63–72 (2009)
8. Gropp, W., Lusk, E.L.: Reproducible measurements of mpi performance characteristics. In: Proc. of the 6th Europ. PVM/MPI Users' Group Mtg. pp. 11–18 (1999)
9. Hoefler, T., Janisch, R., Rehm, W.: Parallel scaling of Teter's minimization for Ab Initio calculations (11 2006), HPC Nano'06 in conjunction with the Intl. Conference on High Performance Computing, Networking, Storage and Analysis, SC06
10. Hoefler, T., Schneider, T., Lumsdaine, A.: Multistage Switches are not Crossbars: Effects of Static Routing in High-Performance Networks. In: Proc. of IEEE Intl. Conf. on Cluster Computing. IEEE Computer Society (Oct 2008)
11. Hoefler, T., Cerquetti, L., Mehlan, T., Mietke, F., Rehm, W.: A practical approach to the rating of barrier algorithms using the LogP model and Open MPI. In: Proc. of the Intl. Conf. on Parallel Proc. Workshops (ICPP'05). pp. 562–569 (June 2005)
12. Ino, F., Fujimoto, N., Hagihara, K.: LogGPS: A Parallel Computational Model for Synchronization Analysis. In: PPoPP '01: Proc. of ACM SIGPLAN symposium on Principles and practices of parallel programming. pp. 133–142 (2001)
13. Jia, B.: Process cooperation in multiple message broadcast. Parallel Computing 35(12), 572–580 (2009)
14. Lam, C.C., Huang, C.H., Sadayappan, P.: Optimal algorithms for all-to-all personalized communication on rings and two dimensional tori. J. Parallel Distrib. Comput. 43(1), 3–13
15. Martínez, D.R., Cabaleiro, J.C., Pena, T.F., Rivera, F.F., Blanco, V.: Accurate analytical performance model of communications in MPI applications. In: 23rd IEEE Intl. Symp. on Parallel and Distributed Processing (IPDPS). pp. 1–8 (2009)
16. Moritz, C.A., Frank, M.: LoGPC: Modeling network contention in message-passing programs. IEEE Trans. on Par. and Distrib. Systems 12(4), 404–415 (2001)
17. Mudalige, G.R., Vernon, M.K., Jarvis, S.A.: A plug-and-play model for evaluating wavefront computations on parallel architectures. In: IEEE International Symposium on Parallel and Distributed Processing. pp. 1–14 (2008)
18. Pjesivac-Grbovic, J., Angskun, T., Bosilca, G., Fagg, G.E., Gabriel, E., Dongarra, J.J.: Performance Analysis of MPI Collective Operations. In: 4th Intl. Workshop on Perf. Modeling, Evaluation, and Optimization of Par. and Distrib. Syst. (2005)
19. Rodríguez, G., Badia, R.M., Labarta, J.: Generation of simple analytical models for message passing applications. In: Euro-Par 2004 Parallel Processing. Lecture Notes in Computer Science, vol. 3149, pp. 183–188 (2004)
20. Touri no, J., Doallo, R.: Performance evaluation and modeling of the Fujitsu AP3000 message-passing libraries. In: 5th Intl. Euro-Par Conf. pp. 183–187 (1999)
21. Träff, J.L., Ripke, A.: Optimal broadcast for fully connected processor-node networks. Journal of Parallel and Distributed Computing 68(7), 887–901 (2008)
22. Xu, Z., Hwang, K.: Modeling communication overhead: MPI and MPL performance on the IBM SP2. IEEE Parallel Distrib. Technol. 4(1), 9–23 (1996)