# Optimizing Communication Overlap for High-Speed Networks

Costin Iancu
Lawrence Berkeley National Laboratory
cciancu@lbl.gov

Erich Strohmaier
Lawrence Berkeley National Laboratory
estrohmaier@lbl.gov

## Abstract

*Modern networking hardware supports true non-blocking communication and effective exploitation of this feature can lead to significant application performance improvements. We believe that algorithm design and optimization techniques that hide latency by taking advantage of communication overlap will facilitate obtaining good parallel efficiency and performance on the highly concurrent contemporary systems. Finding an optimal, performance portable implementation when using non-blocking communication primitives is non-trivial and intimidating to many application developers. In this paper we present a methodology for discovering optimal message sizes and schedules for a variety of application scenarios. This is achieved by combining an analytic model that takes into account the variability of performance parameters with system scale and load with heuristics designed to avoid network congestion. We perform experiments to understand network behavior in the presence of overlap and purge the optimization space for any system based on either resource or implementation constraints. Our approach is able to choose optimal or nearly optimal implementation parameters for a variety of highly non-trivial scenarios and networks with different performance characteristics. Implementations based on parameters chosen by the models are able to hide over* $90\%$ *of communication overhead in all cases.*

***Categories and Subject Descriptors*** C.4 [*Performance of Systems*]: [Design studies, Modeling techniques]; I.6.4 [*Computing Methodologies*]: Simulation and Modeling—Model Validation and Analysis

***General Terms*** Measurement, Performance, Design

***Keywords*** Communication overlap, Latency hiding, Performance model, High-speed networks, Variability, LogGP, LogP

## 1. Introduction

A large number of application and networking performance studies [3, 6, 9, 12, 13, 20] has shown considerable performance benefits when using non-blocking communication primitives and exploiting communication computation overlap. We believe that due to the levels of concurrencies proposed for Petascale systems, efficient use of non-blocking communication including overlapping will be one of the keys for achieving good performance for many scientific applications.



*PPoPP'07* March 14–17, 2007, San Jose, California, USA.


Optimizing applications to hide communication overhead requires careful decomposition and scheduling of their communication and computation phases. In general, when compared with "legacy" MPI applications, such optimized programs tend to perform a larger number of relatively finer grained communication operations. The increased performance comes with a price: it is a lot harder to assess the optimal implementation choices at the application level. This is due to the large space of possible program optimizations as well as the complex interactions between the system components (processor, network) and the highly unpredictable behavior of the system when congestion at any level is present. The studies presented by Danalis et al. [9] and Koziris et al. [12] show the benefits of using non-blocking communication and overlap for micro-benchmarks and applications written in MPI but fall short of providing a methodology for choosing optimal implementation parameters.

Previous work in the area of application and network performance modeling has disregarded the variation of network performance parameters with system workload, scale, application communication schedule and pattern. This variation matters and directly impacts performance.

In this paper we introduce a methodology for quickly determining the optimal decompositions and schedules for applications that use non-blocking communication. We incorporate the information about network performance variability with system scale and workload into a framework that takes into account several application characteristics such as: *message size*, *communication schedule* and *communication pattern*. Using our approach we can correctly indicate how to organize code to achieve maximal overlap for communication patterns widely encountered in applications.

One distinguishing characteristic of our work is that we do not attempt to build a performance model for accurately predicting the end-to-end execution time. Instead, we build a model to capture the behavior of the optimal implementation as well as the imposed limitations from hardware and software constraints. We use the output of our models to guide the search for values for the decomposition granularity of computation and the pipelining of communication that achieve optimal performance.

This paper makes the following contributions:

- An evaluation of the overlap potential present in a system and how it can best be exploited.
- An understanding of how hardware resource constraints influence performance and restrict implementation choices.
- A systematic methodology for evaluating and characterizing communication patterns.
- An analytical performance model which captures the functional dependency on all first-order variables of different implementations of computation and communication overlap.

To our knowledge, this work is the first to propose a systematic solution for choosing performance portable application de-

compositions for optimal overlap in parallel applications. Using our approach, we were able to automatically select implementation parameters that hide over 90% of communication overhead in non-trivial application scenarios. We believe that the work presented here is of interest to application developers, compiler writers, and developers of automatically tuning libraries. This approach is currently implemented in the loop optimization framework of the Berkeley UPC compiler.

Furthermore, we believe that obtaining good performance on Petascale systems will be highly dependent on performance optimizations which consider hardware resource constraints and our work presents some guidelines on how to achieve this.

In the rest of this paper, for brevity, we will frame our discussion in terms of a one-sided communication model and API: communication operations are initiated by an *init()* call and are finished by a *sync()* call. However, the implementation templates we examine have a directly corresponding MPI implementation and the methodology is able to capture the performance characteristics of MPI programs.

In Section 2 we describe our general methodology, the basis of our performance models, and give a systematic description of all parameters we consider for achieving computation and communication overlap. Section 3 presents our experimental setup, and sections 4-7 discuss in detail our findings for four prominent classes of communication patterns. In Section 8 we discuss the sensitivity of the optimal solutions to our parameters and to other potential sources of errors to demonstrate the robustness of the presented approach. The final two sections discuss related work and present a short summary.

## 2. Methodology

Network performance models have been traditionally used to explore the design space of parallel applications. There is a large body of work that uses LogP [8] or LogGP [1] to motivate choices between implementation alternatives. All these approaches start by building an accurate end-to-end timing model for the applications of choice and validating the accuracy of the model. Only after the model is deemed "time accurate" can implementation choices be justified. The models we have seen in the literature require a very detailed instrumentation and understanding of the application behavior and are not well suited for fast prototyping. Another common characteristic of most, if not all, application modeling studies we have seen is that they disregard the variability of network performance parameters with system scale and load.

Previous studies of application behavior [18, 24] indicate that scientific applications tend to have a high communication volume. In the presence of such high communication volumes, the accuracy of traditional time accurate models becomes very sensitive to changes in network bandwidth or latency due to various sources of congestion [1]. For example, MPI studies [6, 22] have shown the performance impact of long message queues on latency. These variations increase with system size or load. Furthermore, we expect modern torus based network architectures (Cray XT3, BG/L) to show an even larger variance of performance parameters than the current fat-tree networks.

Our stance is that in order to be able to choose implementation parameters that are able to provide optimal overlap across system scales and problem settings, one needs to take into account the quality of service the system is able to provide for that given setting. We achieve this by exploring network performance in terms of system scale and hardware resource constraints (e.g. outstanding message queue depths) and we identify the situations where performance degrades due to any form of congestion.

To determine optimal overlap parameters we characterize programs according to several metrics: 1) communication computation ratio; 2) volume of communication and 3) communication patterns.

For any given combination we are interested in determining the implementation parameters (*communication granularity* and *schedule*) that minimize the end-to-end running time or alternately, maximize overlap while staying away from situations where congestion might occur. We consider this as the $optimal$ implementation. For each combination of our key metrics, we build a model to capture the performance for the $optimal$ implementation case and identify the constraints imposed by it, e.g. number of messages and message injection rate. These constraints allow us to efficiently purge the design space of choices that are likely to cause congestion. We can then search between the remaining implementation alternatives and pick the one likely to offer best behavior based on the performance model. The approach is specifically designed for on-line (runtime) optimizations and therefore one of the main goals is to produce simple enough models that allow a fast evaluation and pruning of the solution space.

In order to reduce the initial exploration overhead, we characterize the behavior of network performance parameters using a very discrete sampling. Searches through the solution space for any given scenario are also performed in a discrete manner. Thus, throughout this paper we consider the optimal implementation choice to be the point in the discrete parameter space that we have explored that offers best performance compared to the other visited points. Using the mathematical connotation, an optimal solution will provide the absolute minimum of the end-to-end running time for a given scenario. We believe that our approach is also able to determine this solution, but it requires either a complete exploration of the optimization space or characterization of network performance using continuous functions. This is a limitation of any performance modeling effort we are aware of and either approach has a very high time to solution.

Our methodology for fast prototyping of implementation choices does not rely on building a time accurate performance model which we believe to be a daunting task at very high concurrency levels and for implementations that aggressively use non-blocking communication. The method can be incrementally refined to capture the performance trends of any implementation choice that resides outside the candidates that satisfy the optimality conditions as long as it does not cross into the regime where hardware resource constraints apply.

### 2.1 Network Performance Characterization

The LogGP [1, 8] network performance model approximates the cost of a data transfer as the sum of the costs incurred by the transfer in all system components. The parameters are $o_s$ and $o_r$, the send and receive overhead of a message; $L$ round-trip network latency; $G$, the inverse network bandwidth; and $g$, the minimal gap required between the transmission of two consecutive messages. In this section we present our usage of the LogGP model and discuss how its parameters vary with system workload and scale.

According to the model, the cost of a single message transfer can be divided into two components, the software overhead on both the send and receive side, and the time the message actually spends in the network. The total communication time for a message of size $S\ bytes$ is

$$T(S) = o + L + G * S, \qquad o = o_s + o_r$$

In the ideal case, a fraction equal to $L+G*S$ from the total message communication time can be overlapped with independent work.

[1] By congestion we denote not only bandwidth degradation due to packets being dropped inside switches, but any other decrease in response time due to resource exhaustion, e.g. overflowing the hardware or software outstanding messages queue length.

| System | Network | CPU type |
|---|---|---|
| AMD cluster [11] | Infiniband | 640x 2.2GHz Opteron |
| AMD cluster | Elan4 | 16x 2.2 Ghz Opteron |
| Alphaserver ES45 [15] | Elan3 | 3000x 1 GHz Alpha |

**Table 1.** Systems Used for Benchmarks

To determine the values of the network performance parameters we use the methodology presented in [2] combined with the micro-benchmarks presented in Section 3. We examine network performance on large scale production systems with Infiniband and Quadrics (Elan3) networks and a small cluster with Quadrics (Elan4). The systems are described in Table 1 and the parameters are determined for the GASNet [4] communication layer.

Both Infiniband and Quadrics have Remote Direct Memory Access (RDMA) support. On Infiniband, the memory involved in RDMA operations has to be registered with the card and pinned by the operating system. Quadrics has an on-board TLB which synchronizes with the processor TLB and does not require pinning[2].

Both networks impose hardware constraints on the number of outstanding communication operations allowed to proceed concurrently. We refer to this value as *hardware queue depth* ($HD$). Both networks implement flow of control based on this value. If there are more than $HD$ concurrent messages, when issuing the $HD+1$ message the processor will block until one of the previous transfers has finished. On Quadrics this value is 32 (Elan3 and Elan4) and on Infiniband this value is 64.

**Latency:** We determine the network round-trip latency and bandwidth using the methodology presented in [2]. For Elan3, we measured $L = 7.4\mu s$, for Elan4 we measured $L = 2.4\mu s$ and for the Infiniband network we measured $L = 10\mu s$. When modeling $get$ operations we use the round-trip latency, while the models for $put$ operations use one-way latency.

**Overhead:** Most of the previous network and application performance modeling efforts consider the overhead of message initiation to be independent of the message sizes and use the value determined for the empty transfer. Kielmann et al. [19] propose an extension to the traditional LogP model to take into account the variation of $o$ with the transfer size and introduce a measurement methodology for wide area networks.

Applications optimized for non-blocking communication are likely to issue back-to-back communication operations. We refer to a sequence of non-blocking communication operations without intermittent computation as a "burst" of requests and denote the number of communication operations as "burst length" $b$ and we examine the overhead variation with the burst length. We are also interested in determining the variation of $o$ with the message size $S$.

We measure the transfer initiation overhead value using two methods: 1) the methodology described in [2] for asymptotic values; 2) a micro-benchmark whose structure is presented is Figure 6-E where we measure the average time per initiation operation for bursts of increasing length. Figures 1 and 2 show the evolution of message initiation overhead with message size and burst length. Each line in the graph corresponds to a different burst length.

For all networks, for a fixed burst length $b$, $o$ increases roughly linearly with the transfer size. For all networks, for burst length $b \leq HD$, the value of the average time taken to issue a message within the burst decreases with the increase in the burst length. For small to medium size messages ( $< 128$ kB) this variation is around

[2] It allows both pinned and un-pinned implementations depending on the kernel integration level. Best performance is achieved with synchronized TLBs.

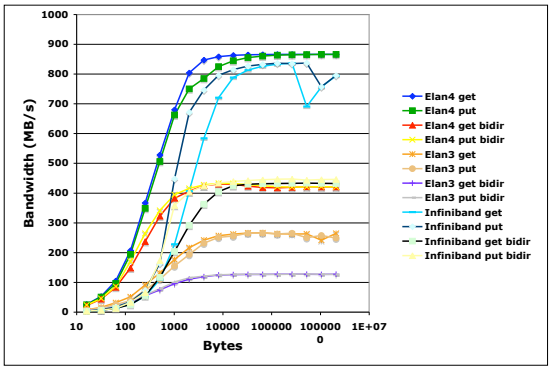


**Figure 3.** Unidirectional and bidirectional **per-processor** peak bandwidth.

30% for short bursts. For larger messages, the variation is as much as 100%. The overhead reaches the asymptotic values determined using the methodology from [2] with the increase in burst length.

Increasing the burst length to values larger than the hardware queue depth $b > HD$, shows the effect of hardware flow control on the overhead of initiating a transfer. The $init$ operation will block and wait for a previous transfer to finish. In our experiments, this effect appears for the Infiniband and Elan4 networks, which observe a sharp increase of the transfer initiation overhead for values of $b > HD$. On these networks, the issue rate (as determined by $o$) is faster than the network service rate as determined by $G$. On Elan3, which has a high initiation overhead, the effect is not observable in the ranges exercised by our benchmark. The best illustration of this effect is shown in Figure2.

For our performance models, we take into account the variation of $o$ with both message size $S$ and burst length $b$. In the rest of the paper we'll use the shorthand $o = o(S, b)$. For a fixed burst length $B$, $o(S, B)$ can be approximated reasonably well by a piecewise linear function on all systems considered. In practice, we use the table containing the experimental values.

**Gap:** In LogP, the $g$ parameter is used to describe any possible idle time of the main processor between the initiation of two successive communication events. Figures 1 and 2 show a decreasing overhead for increasing burst rates. While there is no guarantee that the processor might idle between message initiations and hence a positive $g$ might exist, it is very unlikely, as this would require a an even lower effective value of $o$ than presented. We therefore ignore this parameter for the rest of the paper.

**Bandwidth:** Figure 3 shows the variation of the bandwidth between two active processors for all networks as a function of transfer size. On the measured systems, *per processor* bi-directional bandwidth is roughly half of the unidirectional bandwidth due to PCI bus bandwidth limitations. The network is saturated at a different message size in the uni-directional case than in the bidirectional case.

The production systems we study contain well provisioned fat-tree networks with good bisection bandwidth. However, in application settings it is often the case that full bisection bandwidth is not attained at high concurrency levels, due to the unfairness of bandwidth allocation caused by congestion or the implementation of the communication software layer.

Figure 4 shows the per processor pair bandwidth on the Infiniband network for 128 processors communicating in a nearest neighbor[3] pattern ($P_i - P_{i+1}$). In this case the per-connection bandwidth

[3] Consecutive switch ports.

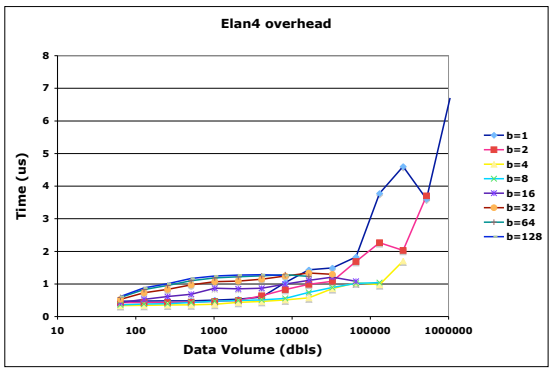


**Figure 1.** Overhead ($o$) on Elan4, **b** denotes burst length, x-axis has logarithmic scale

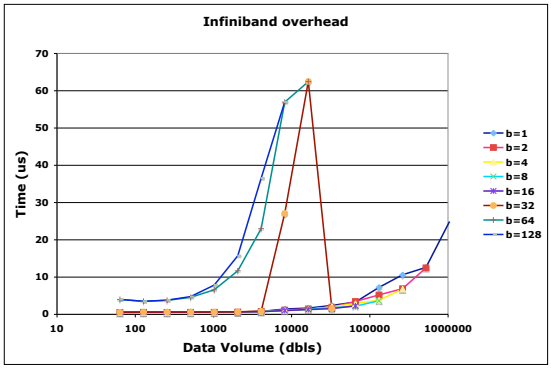


**Figure 2.** Overhead ($o$) on Infiniband, **b** denotes burst length, x-axis has logarithmic scale

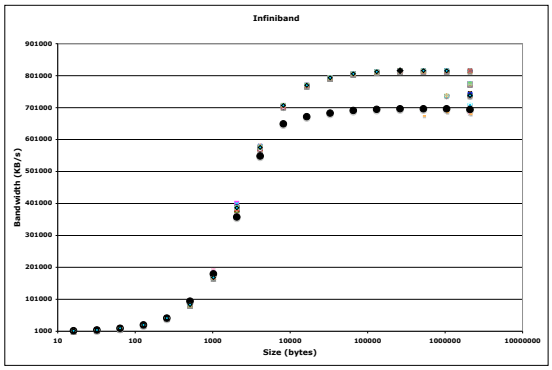


**Figure 4.** Variation of unidirectional bandwidth for Infiniband, 128 processors communicating in a nearest neighbor pattern.

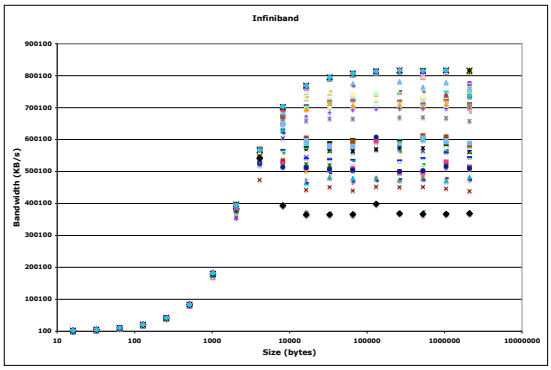


**Figure 5.** Variation of unidirectional bandwidth for Infiniband, 128 processors communicating across the network bisection.

allocation is relatively fair and all processors achieve a bandwidth close to the peak unidirectional bandwidth. Similar behavior is observable on the Elan3 network.

Figure 5, shows the per processor pair bandwidth on the Infiniband network for 128 processors communicating in a cross network pattern ($P_i - P_{i+half}, i < half$). In this case the bandwidth allocation is very unfair with a very large variance between the lowest and the highest bandwidth achieved. Similar behavior is observable on the Elan3 network.

The unfairness of the allocation increases with the degree of concurrency and is typical for static source routed networks. Wormhole routed networks might provide better bandwidth allocations especially for large competing transfers, but we still expect a large variance at very high concurrency levels.

Figures 3, 4, 5 show that bandwidth and implicitly application performance is directly determined by the topology of the communicating processes and system size. For our model we record the bandwidth variation with topology and system size. For each of the two cases (N=nearest-neighbor and C=cross) we record the lowest and highest bandwidth levels achieved by a pair of processors. We denote these values by $G^l_N(P,S), G^h_N(P,S)$ and $G^l_C(P,S), G^h_C(P,S)$ respectively.

### 2.2 Application Characteristics

In this section we present our classification of different application scenarios using a one-sided communication paradigm $init/.../sync$.

The communication operations are either remote read ($get$) or remote write ($put$) operations. Equivalent scenarios can be implemented using the MPI two-sided communication paradigm.

We have explored the performance of a comprehensive set of implementation choices, which are not presented in this paper. The "templates" selected here offer best performance in practice and exhibit a symmetry that reduces the number of tunable parameters and also makes them amenable to manual code transformations. Intuitively, the patterns presented here achieve good performance by obeying common sense design principles: 1) issue communication as early as possible and 2) issue as many as possible consecutive communication operations in order to reduce the initiation overhead.

Figure 6 presents possible implementation scenarios: (A) shows a program using blocking communication; (B) shows a program using communication-communication overlap; (C) shows a program with *tight* data dependencies that overlaps communication with both communication and computation and is usually encountered in MPI programs or $get$ based one-sided implementations; (D) shows a program with *loose* data dependencies that overlaps communication with both communication and computation.

We consider the program schedule to be the sequence of $init/comp/sync$ operations. We use the schedule for the blocking communication scenario (A) as the performance baseline. Implementation (B) has a higher degree of overlap than (A) and can achieve better performance. Both implementations (C) and (D)

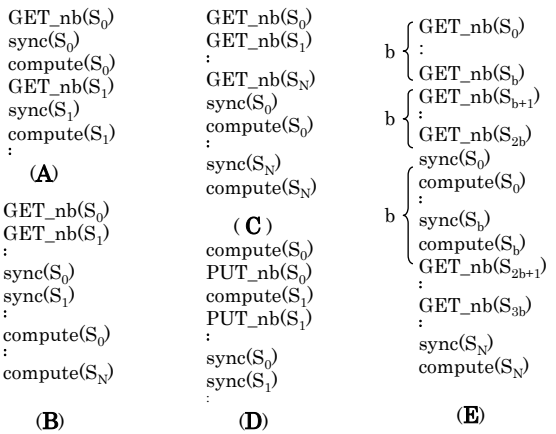


**Figure 6.** (**A**) Blocking communication; (**B**) Communication communication overlap; (**C**) Communication/computation overlap with *tight* data dependencies; (**D**) Communication/computation overlap with *loose* data dependencies; (**E**) Pipelined implementation of (C) with $burst$ size $b$.

have a higher degree of overlap than (B) and have the potential to achieve even better performance. Additional program transformations can be applied to the program schedule to improve the performance of (C). Implementation (E) shows the modified schedule after a technique similar to software pipelining has been applied to the program schedule in (C).

The granularity of the computation present in the application determines how much time is available for overlap and influences the choice of an implementation strategy. We distinguish between the case where the application is *communication bound* and the case where the application is *computation bound*.

Another factor that determines end-to-end performance is the application communication topology. This includes the number of communication end-points and the order in which they are accessed. From the connectivity point of view we distinguish the following cases described throughout the rest of the paper: *point-to-point*, *multiple end-points*, *end-point-congestion*, *all-to-all*.

Oliker et al. [18] study eight large scale scientific applications and report that over $90\%$ of the MPI calls encountered are *point-to-point* operations. Similar trends are reported by Vetter et al. [24] for five different large scale scientific applications. Both studies find that the majority of applications communicate with a relatively low number of partners, regardless of the system size. Some applications change endpoints within the same time step, some of them across time steps. The groups of communicating processes are usually static during the execution. The few applications that exhibit rich connectivity usually perform all-to-all or gather communication calls.

For a given application, let $V$ denote the per-processor data domain size and let $S$ denote the message size after further decomposition for communication. The programmer's interest is to create a schedule of communication and computation operations that takes into account data dependencies and produces optimal performance. In the following, let $N = \frac{V}{S}$ denote the total number of communication operations generated by the decomposition. Given a problem with a total volume of data $V$ we are interested in determining the granularity $S$ and the burst length $b$ that is likely to offer best performance.

We examine implementations (C), (D) and (E) since they contain non-trivial communication computation overlap.

## 3. Modeling Communication Performance

We validate the methodology on two contemporary high speed networks: Quadrics and Infiniband. The systems are described in Table 1. The evaluated systems exhibit different network performance characteristics (latency to overhead or inverse bandwidth ratios) and different architectural choices (TLB). Both networks are connection oriented, use source routing and exhibit qualitatively similar responses to congestion. We have started to validate the methodology with similar results for the IBM Federation HPS network which implements RDMA over an unreliable datagram layer and uses adaptive routing. The Infiniband standard provides for similar functionality, currently unsupported by any vendor.

All the benchmarks are written in UPC [23], a Partitioned Global Address Space language (PGAS) which provides one-sided communication abstractions. The UPC implementation runs over GASNet [4], a portable high performance one-sided communication library. Thus, our results capture behavior at the application level, rather than behavior at the communication layer level. This is important since the extra overhead of the runtime/application setting needs to be taken into account when trying to understand whole application performance.

We use synthetic benchmarks to build the performance models and validate the model results. The synthetic benchmarks correspond to the code patterns described in Figure 6(A-E). The benchmark implementation is capable of varying the computation to communication ratio present and to exercise all the patterns mentioned in Section 2.2. For each connectivity scenario (*point-to-point*, *multiple end-points* ...) we have run the patterns (A), (C),(D) and (E) for various concurrency levels (up to 64 processors for most cases, 128 and 256 processors for selected experiments).

As the base computation we have chosen a vector-scalar add operation `b[i] = a[i] + s`. In all cases, this operation produces for mid-size to large messages roughly a $40\% - 50\%$ computation/communication ratio. This is the lowest ratio we have examined. We vary the computation present in the micro-benchmarks up to five times the base ratio, roughly $250\% - 300\%$, and we examine five different settings varying from communication bound to computation bound problems. The total data volume $V$ varies between 8 bytes and 8 MB in powers of two. The burst size $b$ is varied between 1 and 64 in powers of two. For each volume, decompositions of up to 8192 messages are tested in power of two increments.

We have performed this exhaustive exploration of the optimization space in order to validate the methodology. Determining the model parameters and constraints for systems with characteristics similar to the ones presented here *does not* require an extensive set of calibrations. The only values required are profiles for the variation of the $o(S, b)$ and $G(P, S)$ parameters and an understanding of overlap behavior in a point-to-point scenario.

In the paper, we present data only for $get$ operations and the majority of our tests were run with $get$ operations. The results and modeling of $put$ operations are similar, the only differences appear in the values of the parameters that capture the network performance. Since application scenarios with tight dependencies are harder to optimize for, we'll center the following presentation on the scenarios in Figure 6-C and Figure 6-E.

### 3.1 Point to Point Communication

In this section we will discuss the models for the implementations in Figure 6-C and Figure 6-E for communication bound and computation bound problems. The models for the cases in Figure 6-B and 6-D are developed based on the same principles.

**Communication bound:** we consider first the case where the application is communication bound ($G_V * V > t_V$). For the implementation with tight dependencies (Figure 6-C) with a total domain

size $V$, decomposed in sub-domains of size $S$, with a communication schedule with bursts of length $b$, total computation time $t_V$, $N$ communication operations organized in $n_b$ bursts, the *optimal* time for the transformed program is given by Equation (1).

The formula is based on the following assumptions: 1) for each burst, only the latency and overhead of the first communication operation in the burst is not overlappable with useful work; 2) total communication time is determined by the number of messages of size $S$; and 3) the last computation within a burst is not overlapped with anything else.

$$T(V,S,b) = (o+L) * n_b + N * G_S * S + n_b * t_S \quad (1)$$

$$G_V * V - t_V > N * o \quad (2)$$

$$L + G_S * S > (b-1) * o \quad (3)$$

The difference between the communication and computation time ($G_V * V - t_V$) determines the time that the optimized implementation can use to issue the additional communication operations ($N * o$). This imposes the first optimality constraint in Equation (2). In order to achieve perfect overlap, all the communication initiation overhead within a burst should be perfectly hidden by the latency of the first transfer within the burst. This constitutes another optimality constraint in Equation (3).

For the pipelined case (Figure 6-E), assuming perfect overlap, the lower bound on the total running time is given by Equation (4). The formula and the constraints are derived based on the same principles described for the non-pipelined implementation. For balanced problems, where communication time is similar to the computation time ($G_V * V \approx t_V$), the additional communication initiation overhead cannot be hidden any longer. To capture this we use the model in Equation (5) for the total running time. The additional term changes the relative importance of bandwidth and latency when determining the solution based on the computational intensity of the application. Applications that are heavily communication oriented ($t_V << G_V * V$) should be optimized for bandwidth, while for better balanced applications the initiation overhead and network latency matter more.

$$T(V,S,b) = o + L + N * G_s * S + t_S \quad (4)$$

$$T(V,S,b) = (o+L) * n_b * \frac{t_V}{G_V * V} + N * G_s * S + t_S \quad (5)$$

**Computation bound:** for the case where the application is computation bound ($G_V * V < t_V$) the optimal implementation is described by Equation (6) for the non-pipelined (Fig 6-C) case and Equation (7) for the pipelined (Fig 6-E) case .

$$T(V,S,b) = (o + L + G_S * S + b * t_S) * n_b \quad (6)$$

$$T(V,S,b) = (o + L + G_S * S) + b * t_S * n_b + o * b * (n_b - 2) \quad (7)$$

For all cases, an additional implementation constraint introduced to avoid flow of control problems when overflowing the message queues is:

$$b < HD \quad (8)$$

Figures 7 and 9 show the range of decompositions that achieve speedup for a communication bound scenario on the Infiniband and Elan4 networks. The *x-axis* corresponds to the total problem size ($V$) and the *y-axis* corresponds to a given message size ($S$). The results are reported relative to the fastest decomposition which corresponds to the value 1 ($lowest$) in the color-map. The highest value corresponds to the implementation that performs blocking communication and has no overlap. The scatter line labeled $measured$ shows the decomposition that achieves best speedup for a given volume. The scatter line labeled $model$ shows the predictions of our model.

The results indicate that for any given problem size, there exists either an optimal decomposition or a set of decompositions that offer "best" performance and that are statistically indistinguishable. Increasing the burst size improves the performance of finer granularity decompositions. Increasing the problem size determines an increase in the optimal decomposition size. For a given problem setting, due to lower overhead and latency, the Quadrics hardware achieves the best results when using smaller messages than for the Infiniband hardware. Computation bound problems (Figure 8) achieve best performance at finer grained decompositions. Even for this simple point-to-point scenario, performance portability across platforms and input sets is hard to achieve and there is no good static implementation solution.

On Quadrics hardware, the optimal message size and communication schedule (*burst size*) is further limited by the NIC TLB coverage (128 entries on Elan4, 32 entries on Elan3). The network ability to overlap increases with the granularity of the decomposition and the burst size up to the point where TLB coverage is exhausted. Decompositions operating beyond this threshold observe decreased performance. Figure 10 illustrates this best. Thus, for the Quadrics hardware we introduce an additional empirical constraint to our model: we limit the volume of outstanding communication requests to 4MB. This constraint has been determined by inspection of the experimental data across all scenarios and it is important for the accuracy of predictions on Elan (TLB based) networks.

We have examined the model predictions using two estimates for the value of communication initiation overhead $o$: 1) taking into account only the variation with message size $o = o(S)$; and 2) taking into account the variation with both message size and burst length $o = o(S,b)$. For the networks with a relatively high latency (Infiniband and Elan3) the first estimator produces relatively accurate results. For the Elan4 network which has very low latency, the second estimator is required for accurate predictions. We therefore use $o = o(S,b)$ for all cases.

A property of the model is that for a fixed combination (V,S), the total time monotonically decreases with the increase in burst length $b$. Thus our model will always choose the longest burst within constraints. The experimental results validate this choice.

### 3.2 Communication With Multiple Endpoints

Communication topologies with multiple endpoints are very typical for a variety of scientific applications. Examples include grid based algorithms where values along common borders have to be exchanged with multiple processors. Other examples would be scatter-gather or personalized broadcast operations.

In this scenario, one processor exchanges data with $P$ other processors and the ordering of the end-point accesses is an additional factor that determines performance. We call *communication peer sequence* the order with which distinct processors are accessed within a program schedule and we consider two cases: $contiguous$ and $interleaved$. If a processor $P_i$ communicates with two peers $P_j$ and $P_k$, in a contiguous schedule $P_i$'s communication will have the following end-point ordering: $P_j, P_j ... P_k, P_k$; processor $P_i$ finishes all the work for $P_j$ before moving to another processor. In the interleaved pattern, $P_i$ will impose the order $P_j, P_k ... P_j, P_k$; processor $P_i$ alternates between endpoints.

We model the optimal running time for a problem with a *contiguous* schedule as $P$ instances of the point-to-point scenario $T(P,V,S,b) = P * T(V,S,b)$. All the equations have been shown in the previous section.

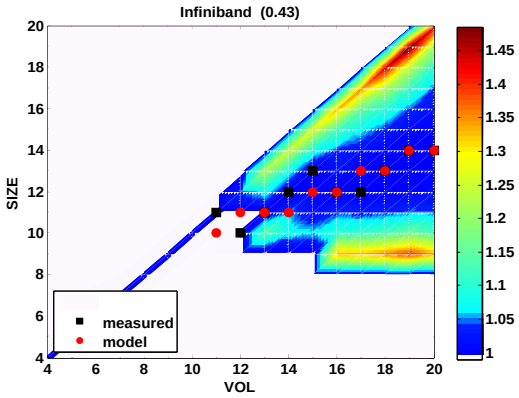


**Figure 7.** Performance of different communication schedules, together with measured and model-predicted best schedules for Infiniband. Axis are labeled with log values. *Point to point communication bound problem:* 0.43 = $\frac{T_{comp}}{T_{comm}}$. VOL=total problem size, SIZE=message size. Colormap shows loss of performance from best decomposition.

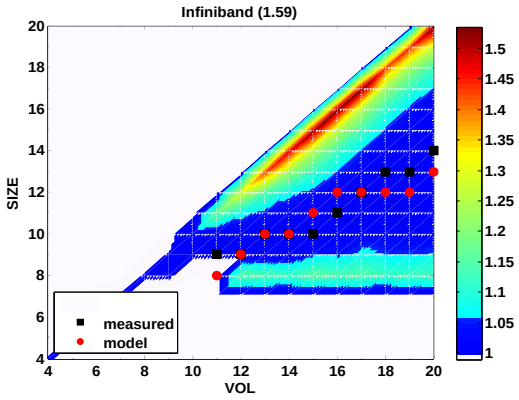


**Figure 8.** Performance of different communication schedules, together with measured and model-predicted best schedules for Infiniband. Axis are labeled with log values. *Point to point computation bound problem:* 1.59 = $\frac{T_{comp}}{T_{comm}}$. VOL=total problem size, SIZE=message size. Colormap shows loss of performance from best decomposition.

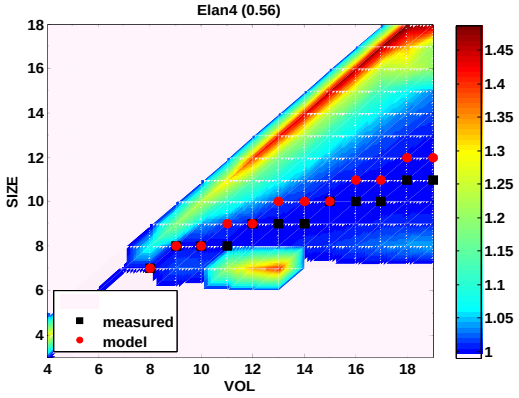


**Figure 9.** Performance of different communication schedules, together with measured and model-predicted best schedules for Elan4. Axis are labeled with log values. *Point to point communication bound problem:* 0.56 = $\frac{T_{comp}}{T_{comm}}$. VOL=total problem size, SIZE=message size. Colormap shows loss of performance from best decomposition.

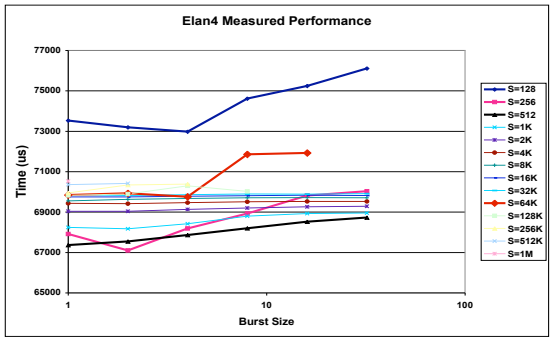


**Figure 10.** Performance with *end-point congestion* on Elan4 (Figure 6-E, 4 threads). Total problem size (V) is 1M doubles. Legend shows the decomposition size. The *knee* in the lines shows the effect of TLB restrictions.

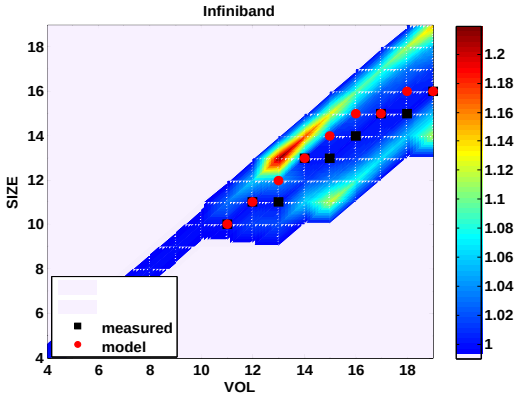


**Figure 11.** Performance of different communication schedules, together with measured and model-predicted best schedules for Infiniband. *Multiple endpoints, contiguous peer schedule, 32 threads.* Axis are labeled with log values. VOL=total problem size, SIZE=message size. Colormap shows loss of performance from best decomposition.

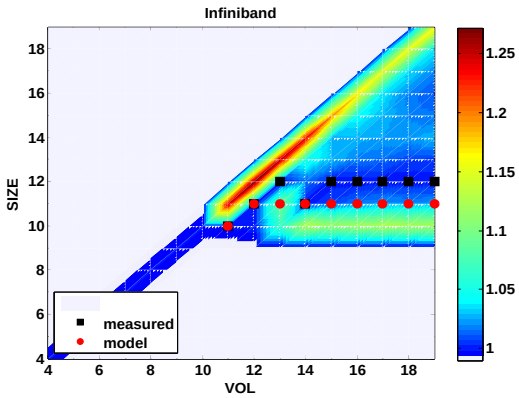


**Figure 12.** Performance of different communication schedules, together with measured and model-predicted best schedules for Infiniband. *Multiple endpoints, interleaved peer schedule, 32 threads.* Axis are labeled with log values. VOL=total problem size, SIZE=message size. Colormap shows loss of performance from best decomposition.

Figures 11 and 12 show all the decompositions able to provide speed-up on the Infiniband network for a 32 processor[4] run. In this case, we consider as a base case for performance comparison the situation where communication with each processor is already overlapped with other communication and computation. Thus the results show the additional benefit of more aggressive optimizations. The optimal decomposition for the contiguous case resembles the point-to-point case. The interleaved case attains best performance at much finer granularity. Similar behavior is observable on the Elan networks.

This difference is explained by the fact that the interleaved schedule produces a higher route contention inside the network switches than the contiguous schedule. Finer granularity messages offer a better interleaving of the communication operations. At the model level, the difference is captured by the choice of the bandwidth profile. The contiguous case generates a lower degree of switch contention and we use for it the lowest bandwidth level determined for a "nearest-neighbor" pattern $G^l_N(P,S)$ (Figure 4) at the respective concurrency level. For the interleaved pattern we use the lowest bandwidth level determined for a "cross" pattern $G^l_C(P,S)$ at the same concurrency. Figures 11 and 12 show the predictions of our models. In all cases we have examined, the models produce good results.

For an implementation, given the choice of the two peer schedules, one legitimate question is which one is likely to perform better. We have collected data and measured additional parameters for an alternate model for the *interleaved* case. A detailed description is beyond the scope of this paper. Using the distinct models, we were able to make accurate (in the majority of cases) qualitative predictions between the two implementations using the same parameter ($P$,$S$,$b$) settings.

The performance data shows that for a given problem setting ($P$,$V$) with the same implementation parameters ($S$,$b$), the *contiguous* schedule is faster than the *interleaved* schedule for most cases. The two implementations achieve optimal performance for different values of $S$ and $b$, and in this case the $interleaved$ schedule is faster. In practice, the differences are minimal for the optimal case.

### 3.3 End Point Congestion

This communication topology appears, for example, in situations where global grids have to be collected and assembled by individual processors or in master-slave programs where slave data needs to be transfered to the master.

In this application scenario, which captures the serialization of concurrent transfers inside the NIC, P processors transfer data to or from the same processor. Figure 10 shows the performance for the scenario from Figure 6-E. The results for this experiment show the same trends as for the previous experiments. On the Quadrics networks, this is the most sensitive pattern to the TLB interference.

The model for the optimal end-to-end time for this case is presented in Equation (9). Each processor performs a point-to-point transfer. The additional term captures the serialization of the communication requests on the congested NIC.

$$T(P,V,S,b) = T(V,S,b) + (P-1) * b * G_S * S \quad (9)$$

Solving the optimization problem produced optimal or nearly optimal values for the optimization parameters.

[4] This is the "first" concurrency level where ignoring system scale effects leads to inaccurate predictions and to potentially bad implementation choices.

### 3.4 All-to-All Communication

In general, for all-to-all communication patterns, implementations need to take congestion into account and need to eliminate communication hot-spots. When using non-blocking point-to-point communication primitives to implement all-to-all patterns, network congestion is determined by the message injection rate ($b$) and the communication peer schedule. The models and considerations presented here are for implementations that avoid hot-spots by performing all-to-all communication in a staggered manner. Stagering communication [10] to avoid hot-spots is a technique widely used in the implementation of collective communication library calls. The application studies [3, 9] report significant performance improvements when replacing all-to-all collective calls with non-blocking overlapped point-to-point communication.

We examine the performance of several all-to-all communication/computation patterns. In the first scenario, all the processors involved in the operation observe the same peer schedule, e.g. $P_0, P_0..P_1, P_1...$ This is the worst case for performance since it has communication hot-spots and transfers are serialized by the congested NICs. We model it as the equivalent of $P$ scenarios with end-point congestion (Equation 9).

With a *contiguous* schedule, a processor $P_i$ communicates with $P_{i+1}, P_{i+1}..P_{i+2}, P_{i+2}..$ in this order. We model this case as a combination of a multiple end-points and end-point congestion scenario. Equation 10 shows the model for a communication bound problem and a pipelined implementation (Fig 6-E). The last term in the equation accounts for any possible load imbalance in the system. Whenever a processor $P_i$ finishes processing the domain from a peer $P_j$ and issues the communication requests for $P_{j+1}$, it is likely that $P_{j+1}$ still has outstanding communication requests from another active peer at that stage. We assume that the amount of outstanding communication is bound by $2 * b$ which seems a reasonable upper limit if communication loads are approximately balanced.

$$\begin{aligned} T(P,V,S,b) &= (o+L) * n_b * \frac{t_V}{G_V * V} + N * G_s * S + t_S \\ &\quad +(P-1) * 2 * b * G_S * S \end{aligned} \quad (10)$$

With an *interleaved* schedule, each processor $P_i$ communicates in order with $P_{i+1}, P_{i+2}..P_{i+1}, P_{i+2}...$ In this case, at any stage, a processor will serve on average $b$ transfers. Equation (11) shows the model.

$$\begin{aligned} T(P,V,S,b) &= (o+L) * n_b * \frac{t_V}{G_V * V} + N * G_s * S + t_S \\ &\quad +n_b * b * G_S * S \end{aligned} \quad (11)$$

In both cases, we choose the values for inverse bandwidth $G$ based on the principles described in Section 3.2. Inspection of the equations (10) and (11) reveals that the minimum occurs for $b = 1$, regardless of the message size $S$. This property is substantially different from other communication topologies such as point to point, where the models tend to maximize $b$ within constraints.

We have examined the three scenarios for concurrencies up to 64 processors and for different communication/computation ratios. We have also tried to distinguish between load balanced and load unbalanced problems. To approximate a load balanced problem we synchronize all processors after each iteration of the respective benchmark. For a load imbalanced problem, we allow the processors to perform multiple iterations between a global synchronization step.

For a communication bound problem, different behavior is observed in practice depending on the system imbalance. For "load balanced" problems, optimal performance is achieved at relatively

fine granularity decompositions and deep burst length, regardless of the processor schedule ($contiguous$ or $interleaved$). The models in equations (10) and (11) choose good values for the decomposition ($S$) but underestimate the value of the burst length. In this case, the model for multiple-endpoints (Section 3.2) is an alternative able to choose a good solution. For a $contiguous$ schedule and a load unbalanced problem, optimal values are observed at high granularity decompositions and short burst length ($b = 1$). In this case the solutions determined by the model are not very good. However, applications usually perform one stage of all-to-all communication followed by processing of the data domain and therefore we expect the behavior of the "load balanced" benchmark to be the case encountered in practice.

For communication/computation balanced problems and for computation bound problems, optimal performance occurs at relatively fine grained decompositions and short burst length, regardless of the overall system load. In this case, our model chooses a good granularity at burst length $b = 1$. In practice, the optimal performance is consistently achieved across all experiments at burst length $b = 2$. The loss of performance for the decomposition chosen by our model is very small in all cases. The MPI performance study in [9] also reports optimal performance at very short burst lengths.

## 4. Model Properties

In compilers or automatic tuning libraries, and in general for optimizations without an instrumentation and feedback loop, the optimizer needs to work with relatively coarse approximations of the model parameters. In this section we examine the solution sensitivity and the practical implications of using coarse or wrong estimates for the problem parameters.

One of the parameters most likely to have a coarse estimation is the computation time of the application. For example, compilers might make static estimates of the granularity of a loop nest that requires communication.

**Sensitivity to the estimation of computation time:** for each set of experimental results, we vary the value of the computation time estimate in increments of 10% and record the deviation that results in a change of solution. Sample values for selected scenarios are shown in the first 2 lines in Table 4. The entries labeled "*" indicate that the model always predicts the same values. As a particular example, for a problem with a total size of $4K$ doubles, a $20\%$ overestimation of computation time determines a change of solution.

Across all experiments, communication bound problems are the most sensitive to bad computation time estimates. There's no observable trend in the sensitivity of the respective models. In all cases, using the values for the new solution in an implementation, results in running times close to the optimal time. The models for computation bound problems are very insensitive to the misprediction of the computation time and we do not show any data for these scenarios.

**Model performance and choosing the wrong model:** misprediction of computation time could also result in a choice of the wrong model for a particular problem, e.g. using a model for a computation bound problem while in reality the problem is communication bound. The second part of Table 4 shows the loss of performance due to using the predicted solutions versus the optimal solution and the loss of performance when misdiagnosing the problem type.

We compute the loss of performance as $\frac{T_{model} - T_{opt}}{T_{base}}$, where $T_{model}$ is the observed time when using the model solutions, $T_{opt}$ is the optimal time observed across all experiments and $T_{base}$ is the time for the unoptimized problem. We show sample loss of performance only for selected problems. An entry containing "*" denotes either performance deviations under $1\%$ or an exact match. The errors across all other scenarios have the same magnitude.

For all computation bound cases, the loss of performance due to model solutions is very small ($< 1\%$) and the loss of performance due to misdiagnosing the problem is also small. We do not show data for these cases.

Examining figures 7,8,9, 11,12 in conjunction with the entries labeled "Model" in Table 4 gives an overall idea of the optimality of the solutions chosen by the models. In general, the loss of performance relative to the best solution is relatively small. In all cases, the solutions predicted by the model are able to hide over $90\%$ of the communication overhead. The lines labeled "Misdiagnose" in Table 4 show that for some situations, loss of performance due to the wrong choice of the model is relatively significant. This happens for small to medium transfers and using statically the model for communication bound problems results in good solutions regardless of the problem type.

In general, our results show that in practice is preferable to use an underestimation of the computation time present in the application, rather than using an overestimate. Furthermore, using only the model for communication bound problems (Equation 5) produces good solutions even for computation bound problems regardless of the message size.

**Choosing the correct bandwidth profile:** for all the instances where processors communicate simultaneously with multiple-endpoints (Sections 3.2 and 3.4), the model solution is determined by the choice of the bandwidth profile for the system. We make the assumption that a $contiguous$ peer schedule produces less network contention than an $interleaved$ schedule and accordingly we use different bandwidth profiles for the different scenarios. This assumption holds in practice for all $contiguous$ schedules, regardless of the communication /computation ratio present in the benchmark.

For an $interleaved$ schedule the assumption holds for communication bound problems. Message injection rate is determined by the choice of the burst length (which we control through the model solution) and by the communication/computation ratio. The message injection rate determines directly the degree of network congestion. Communication/computation balanced applications with an $interleaved$ schedule generate a lower degree of contention and the bandwidth profile used in the model need to be chosen accordingly. At the time of this writing, we have collected results for scenarios with a very discrete variation of the computation/communication ratio, e.g. 50% and 100%. We need to further explore the application space between these two ratios in order to better understand the heuristics involved in choosing the right bandwidth profile. Also, note that in this case, the imprecision of compute time estimation is likely to most affect the solution.

Throughout this paper, we have made the distinction between bandwidth profiles determined for a *nearest-neighbor* communication pattern and a *cross-network* communication pattern. In practice, at the concurrency levels we have examined, the bandwidth profile for *nearest-neighbor* it is not needed. Same results are obtained by using the best bandwidth $G_C^h(P, S)$ achieved in the *cross* pattern. We believe this property also holds at very high concurrency levels.

## 5. Related Work

There exists a large number of network performance models [1, 8, 17] and network performance characterization studies. In this paper we use some of the methodology presented by Bell at al. [2] to determine the LogGP performance parameters. Brightwell et al. [6]

| Problem Size (V) | 2K | 4K | 8K | 16K | 32K | 64K | 128K | 256K | 512K | 1M |
|---|---|---|---|---|---|---|---|---|---|---|
| (**A**) Sensitivity to computation estimate | | | | | | | | | | |
| P2P,overestimate | 190% | 20% | 130% | 170% | 50% | 220% | 80% | 130% | 50% | 110% |
| P2P,underestimate | * | * | 80% | 50% | 80% | 50% | 80% | 50% | 90% | 60% |
| (**B**) Errors when misdiagnosing the problem | | | | | | | | | | |
| P2P, Model | 12% | 5% | * | 4% | * | * | * | * | * | * |
| P2P,Misdiagnose | 48% | 22% | 12% | 6% | 1% | * | * | * | * | * |
| A2A,32P,Model | * | * | 6% | 2% | 5% | 2% | * | 1% | 1% | 6% |
| A2A,32P,Misdiagnose | 5% | * | 6% | 2% | 3% | 4% | 8% | 7% | 5% | 6% |

**Table 2.** (**A**) Sensitivity of model to coarse estimation of computation time. Entries represent the imprecision in the computation time estimate that determines a change of solution; (**B**) Loss of performance due to model solutions and misdiagnosing the problem setting on Infiniband for a communication bound problem. P2P - point-to-point, A2A - all-to-all, 32 procs.

present a comprehensive discussion of the tradeoffs for achieving overlap in MPI implementations.

Liu et al. [16] present a performance comparison of the MVAPICH implementation on Infiniband and Elan3 networks. They evaluate the overhead of initiating communication and the overlap potential. They also evaluate the impact of MPI buffer reuse on the performance parameters and report sharp performance degradation on Elan3 when reuse is minimal. Brightwell et al. [5] present a performance comparison of Infiniband and Elan-4 networks using MPI as the communication library. The Infiniband MPI implementation they evaluate does not provide good overlap or communication offload, while the Elan-4 implementation does. They conclude that Elan4 has better scaling characteristics and efficiency. Our micro-benchmark evaluation of the same networks for very aggressive non-blocking communication indicates a reverse trend. Despite the lower latency and higher bandwidth on Elan networks, when using non-blocking communication, scalability is affected by the small TLB size and limited memory footprint. Large OS page sizes might alleviate part of the problem, but their impact on application performance [7] is hard to quantify.

Research into optimizing all-to-all collective communication [10, 14, 21] indicates that achieving optimal performance requires careful scheduling and throttling of the communication operations. Wu et al. [25] study the tuning and characterization of mixed-mode collective operations. They present a detailed performance model for broadcast operations and derive equations to capture the best case, average case and worst case regimen. For a given problem setting only one of the three models offers the right answer and they do not show guidelines for choosing the right model.

Danalis et al. [9] present program transformations for communication computation overlap for MPI programs. They present results for a magneto-hydrodynamic turbulence through spectral methods application ($magneto$) and a viscoelastic turbulent flow in a channel simulation ($visco$). They report similar trends for N-body problems and a molecular dynamics simulation code. They replace MPI_ALLTOALL library calls with point-to-point communication primitives and tiling transformations designed to achieve good overlap. They show experimental results for small system configurations and report the need for a methodology to determine optimal implementation parameters: tile size ($S$) and pipeline depth ($b$). Koziris et al. [12] present a methodology for choosing a pipelined schedule that minimizes completion time for loop tiling with overlap transformation. The also present results for small scale systems and do not have a very good methodology for choosing tile granularities. Our approach is applicable in both cases.

## 6. Conclusion

We have presented a methodology for fast evaluation of implementation parameters that lead to optimal behavior in applications using non-blocking communication and overlap. Our approach is able to provide performance portable implementations that hide over $90\%$ of communication overhead across a variety of non-trivial cases. We do believe that in order to achieve good performance on systems with large concurrency levels, sound design practices that ensure performance portability need to be followed. Performance portability is achieved by incorporating the variance of the system quality of service with scale into a performance model and by carefully throttling communication issue rate to avoid congestion at any level. The scenarios that we explore present some of these techniques.

This evaluation work leads us to believe that developing accurate timing models and using them for performance predictions for applications that non-trivially use non-blocking communication is a daunting task. Performance is determined by a highly multi-dimensional parameter space with non-linear behavior. Our simplified models are accurate enough to predict nearly optimal implementation parameters in a robust way. We require bandwidth profiles for various system concurrencies and an exploration of achievable overlap in a point-to-point (2 processors) setting. We believe the amount of this off-line tuning is easily manageable.

Some of our findings are of direct interest to application developers, while others will be of more interest to compiler writers and developers of automatically tuning libraries. We have applied with good results the models presented here to UPC implementations of the NAS Parallel Benchmarks (MG,CG,FT,IS) and a parallel triangulation application. We are currently implementing the methodology in the Berkeley UPC compiler to guide strip-mining transformations and communication scheduling for deep loop nests that require communication. Parallel libraries such as PBLAS could employ our approach for tuning for optimal performance.

## References


[1] A. Alexandrov, M. F. Ionescu, K. E. Schauser, and C. Scheiman. LogGP: Incorporating Long Messages into the LogP Model for Parallel Computation. *Journal of Parallel and Distributed Computing*, 44(1):71–79, 1997.

[2] C. Bell, D. Bonachea, Y. Cote, J. Duell, P. Husbands, P. Hargrove, C. Iancu, M. Welcome, and K. Yelick. An Evaluation of Current High-Performance Networks. In *Proceedings of 17th International Parallel and Distributed Processing Symposium (IPDPS)*, 2003.

[3] C. Bell, D. Bonachea, R. Nishtala, and K. Yelick. Optimizing Bandwidth Limited Problems Using One-Sided Communication and Overlap. *Proceedings of the 20th International Parallel and Distributed Processing Symposium (IPDPS)*, 2006.

[4] D. Bonachea. GASNet Specification, v1.1. Technical Report CSD-02-1207, University of California at Berkeley, October 2002.

[5] R. Brightwell, D. Doerfler, and K. Underwood. A Comparison of 4X InfiniBand and Quadrics Elan-4 Technologies. *2004 IEEE International Conference on Cluster Computing*, Sept 2004.

[6] R. Brightwell, R. Riesen, and K. D. Underwood. Analyzing the Impact of Overlap, Offload, and Independent Progress for Message Passing Interface Applications. *International Journal of High Performance Computing Applications*, May 2005.

[7] C. Cascaval, E. Duesterwald, P. F. Sweeney, and R. W. Wisniewski. Multiple Page Size Modeling and Optimization. *Proceedings of IEEE Parallel Architectures and Compilation Techniques (PACT)*, 2005.

[8] D. E. Culler, R. M. Karp, D. A. 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.

[9] A. Danalis, K.-Y. Kim, L. Pollock, and M. Swany. Transformations to Parallel Codes for Communication-Computation Overlap. In *Proceedings of the 2005 ACM/IEEE Conference on Supercomputing (SC05)*, page 58, 2005.

[10] D.Roweth and A.Moody. Performance of All-to-All on QsNetII. Available at http://www.quadrics.net.

[11] Jacquard AMD Opteron cluster. LBNL National Energy Research Supercomputing Center.

[12] N. Koziris, A. Sotiropoulos, and G. I. Goumas. A Pipelined Schedule to Minimize Completion Time for Loop Tiling with Computation and Communication Overlapping. *Journal of Parallel and Distributed Computing*, 63(11):1138–1151, 2003.

[13] M. Krishnan and J. Nieplocha. Optimizing Performance on Linux Clusters Using Advanced Communication Protocols: Achieving Over 10 Teraflops on a 8.6 Teraflops Linpack-Rated Linux Cluster. *Proceedings of the 6th International Conference on Linux clusters: The HPC Revolution*, 2005.

[14] S. Kumar and L. V. Kale. Scaling All-to-All Multicast on Fat-tree Networks. In *ICPADS '04: Proceedings of the Parallel and Distributed Systems, Tenth International Conference on (ICPADS'04)*, page 205, 2004.

[15] Lemieux. http://www.psc.edu/machines/tcs/lemieux.html.

[16] J. Liu, B. Chandrasekaran, J. Jiang, S. Kini, W. Yu, D. Buntinas, P. Wyckoff, and D. Panda. Performance Comparison of MPI Implementations over InfiniBand Myrinet and Quadrics. In *Proceedings of the 2003 ACM/IEEE Conference on Supercomputing (SC03)*, 2003.

[17] C. A. Moritz and M. I. Frank. LoGPC: Modeling Network Contention in Message-Passing Programs. In *Proceedings of 2002 IEEE Transactions on Parallel and Distributed Systems*, volume 12-4, April 2001.

[18] L. Oliker, S. Kamil, J.Shalf, and D. Skinner. Understanding Ultra-Scale Application Communication Requirements. *IEEE International Symposium on Workload Characterization (IISWC)*, 2005.

[19] H. E. B. Thilo Kielmann and K. Verstoep. Fast Measurement of LogP Parameters for Message Passing Platforms. *In 4th Workshop on Runtime Systems for Parallel Programming (RTSPP)*, 2000.

[20] V. Tipparaju, M. Krishnan, J. Nieplocha, and D. P. G. Santhanaraman. Exploiting Non-blocking Remote Memory Access Communication in Scientific Benchmarks. *Proceedings of the 2003 International Conference on High Performance Computng, HiPC'2003*, 2003.

[21] V. Tipparaju and J. Nieplocha. Optimizing All-to-All Collective Communication by Exploiting Concurrency in Modern Networks. *Proceedings of the 2005 ACM/IEEE Conference on Supercomputing*, 2005.

[22] K. D. Underwood and R. Brightwell. The Impact of MPI Queue Usage on Message Latency. In *The 2004 International Conference on Parallel Processing (ICPP-04)*, pages 152–160, 2004.

[23] UPC Language Specification, Version 1.0. Available at http://upc.gwu.edu.

[24] J. Vetter and F. Mueller. Communication Characteristics of Large-Scale Scientific Applications for Contemporary Cluster Architectures. *Proceedings of the 2002 International Parallel and Distributed Processing Symposium (IPDPS)*, 2002.

[25] M.-S. Wu, R. A. Kendall, K. Wright, and Z. Zhang. Performance Modeling and Tuning Strategies of Mixed Mode Collective Communications. *Proceedings of the 2005 ACM/IEEE Conference on Supercomputing*, 2005.