Exploiting Communication Latency Hiding for Parallel Network Computing: Model and Analysis

Volker Strumpen and Thomas L. Casavant*
email: strumpen@inf.ethz.ch; tomc@eng.uiowa.edu
Institute for Scientific Computing
ETH Zürich
CH-8092 Zürich, Switzerland

Abstract

Very large problems with high resource requirements of both computation and communication could be tackled with large numbers of workstations. However, for LAN-based networks, contention becomes a limiting factor, whereas latency appears to limit communication for WAN-based networks, nominally the Internet. In this paper, we describe a model to analyze the gain of communication latency hiding by overlapping computation and communication. This model illustrates the limitations and opportunities of communication latency hiding for improving speedup of parallel computations that can be structured appropriately. Experiments show that latency hiding techniques increase the feasibility of parallel computing in high-latency networks of workstations across the Internet as well as in multiprocessor systems.

1 Introduction

Currently, an increasing acceptance of network computing can be observed [1, 2, 3, 4] among scientists and engineers. We define network computing to be utilization of some software platform to support distributed parallel computing in a heterogeneous computer network, usually consisting of a larger number of workstations – both on a local area network (LAN) and in long-haul (WAN) nets. PVM [3] is often considered a de-facto standard in this field, and MPI [5] is emerging as a widely accepted standard. However, running very large problems with high resource requirements of both computation and communication is only beginning to be understood. Furthermore, most systems to-date represent extremely complex implementations, and full understanding of the overheads, capacities, and granularities attendant in such systems is lacking.

*This work was conducted, in part, while the author was on leave from the University of Iowa.

In this paper, we address the communication bottleneck in network environments. For communication performance in LAN-based networks, contention appears to become a limiting factor, whereas latency usually limits communication for WAN-based networks. Latency hiding techniques have been developed to ameliorate this situation. We focus on modeling and analyzing a generic latency hiding technique to be utilized by algorithms that allow for overlapping computation and communication. We define *communication latency hiding* informally as a technique to increase processor utilization by transferring data via the network while continuing with the computation at the same time. To analyze the opportunities and limitations of the technique described in this paper, we:

1. Propose a model for analysis of communication latency hiding by overlapping computation and communication in high-latency networks, and draw conclusions about the possibility to achieve a maximal gain from this overlap.
2. Present some preliminary experimental results and show that our model matches reality well enough to serve as a basis for qualitative prediction of latency hiding performance.

The field of scientific computing provides a large class of applications for which network computing can be effective. In this paper, we begin with a typical technique found in many parallel implementations of scientific applications, domain decomposition, then provide a simple model of gain due to exploitation of communication latency hiding. Finally, a generic computational load measure is introduced that models a much broader class of algorithms. The theoretical analysis has motivated the development of a latency hiding protocol as a refinement of the TCP/IP suite [6]. This refinement is beyond the scope of this paper and is described in [7]. By utilizing otherwise-wasted protocol wait times for computational tasks, latencies are reduced further, and thus smaller granularities can be supported.

2 A Model of Latency Hiding

The performance gain due to communication latency hiding can be captured by means of a simple model. First consider a simple example from scientific computing; the algorithm for an explicit partial differential equation solver based on a five-point-stencil. For parallelization, the spatial problem domain may be decomposed. Artificial boundary conditions are introduced that are exchanged between subsequent iterations. This algorithm has been studied in detail on a LAN of workstations in [4].

2.1 A Simple Model

The purpose of the following model is a simple formalization of the phenomenon of communication latency hiding, and its relationship to problem size and granularity. Assuming a runtime system that supports non-blocking send, and blocking receive, operations, the following SPMD-styled loop body of the solver iteration above would be straight-forward to implement:

(i) Calculate all grid values
(ii) Send artificial boundaries to neighbors $(A.1)$
(iii) Receive artificial boundaries from neighbors

This loop body is separated into two distinct phases, the calculation phase comprises step (i), and the communication phase consists of phases (ii) and (iii). A simple model estimates the execution time t of one loop iteration:

$$t = C * n * m,$$

where n is the x-dimension and m the y-dimension of the problem domain, and C is an application and machine dependent constant that denotes the average calculation time per grid point. Assuming an equal distribution of work among p processors, the calculation time per task is $t_{calc} = t/p$, and we approximate the runtime of the parallelized loop body execution by

$$t_p(p) = \frac{t}{p} + t_{com} = t_{calc} + t_{com}.$$

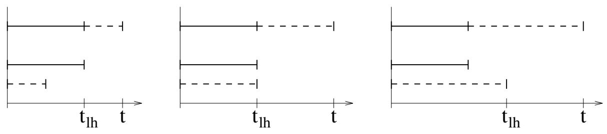
The only overhead incurred is assumed to be communication for this simple model. The speedup of $(A.1)$ is

$$S(p) = \frac{t}{t_p} = \frac{t}{t/p + t_{com}} = \frac{p}{1 + t_{com}/t_{calc}}.$$

Introducing the ratio of calculation to communication per task $\gamma = \frac{t_{calc}}{t_{com}}$, S can be written

$$S(p) = \frac{p}{1 + 1/\gamma}. \qquad (1)$$

In terms of Amdahl's law, $1/\gamma$ can be interpreted as the sequential part of the program. To avoid its disastrous effect with respect to speedup, γ has to be large, i.e. acceptable speedup requires sufficiently large software granularity. However, rather than restricting γ (and thus restricting ourselves to only large problem sizes) another possibility for reducing the (sequential) communication part is considered. The basic idea is that of *communication latency hiding* as illustrated in Fig. 1.



(a) $t_{calc} > t_{com}$ (b) $t_{calc} = t_{com}$ (c) $t_{calc} < t_{com}$

Figure 1: Communication latency hiding illustrated.

In this figure, the solid lines correspond to calculation time t_{calc}, and the broken lines indicate communication time t_{com}. In all three cases, the upper line shows the serialized execution of calculation followed by communication, as given by the parallelization scheme of algorithm $(A.1)$. The lower two lines illustrate calculation and communication starting at the same time, and occurring independently, thus *hiding* the communication latency concurrent with the calculation. The three cases illustrate different degrees of latency hiding. In (a), the communication is hidden thoroughly, corresponding to $t_{lh} = t_{calc} > t_{com}$. The extreme case is shown in (b), where calculation and communication requirements are equal: $t_{lh} = t_{calc} = t_{com}$. In case (c), communication time is larger than calculation time, and thus cannot be hidden completely: $t_{calc} < t_{com} = t_{lh}$.

These considerations suggest that the performance of algorithm $(A.1)$ may be improved, if communication and calculation are executed concurrently. The loop body of a modified algorithm $(A.2)$ is:

(i) Calculate artificial boundaries of next time step
(ii) Send artificial boundaries to neighbors $(A.2)$
(iii) Calculate grid values (except those of step (i))
(iv) Receive artificial boundaries from neighbors

To model the runtime of $(A.2)$, the overall domain is assumed to be sufficiently large, such that step (i) of $(A.2)$ can be neglected. Then, the idealized situation of Fig. 1 holds, and the parallel runtime is

$$t_p(p) = \max\big(t_{calc}, t_{com}\big).$$

The speedup of $(A.2)$ is derived analogously to $(A.1)$:

$$S_{lh}(p) = \frac{p}{\max(1, 1/\gamma)}. \qquad (2)$$

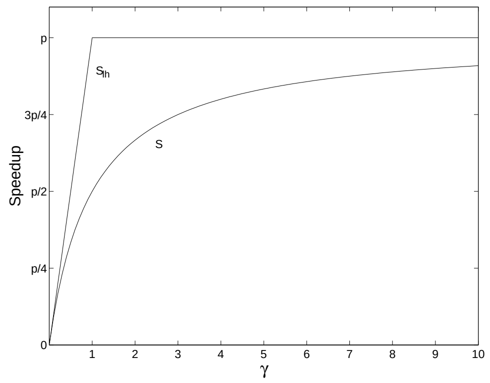


Figure 2: Speedup with and without latency hiding.

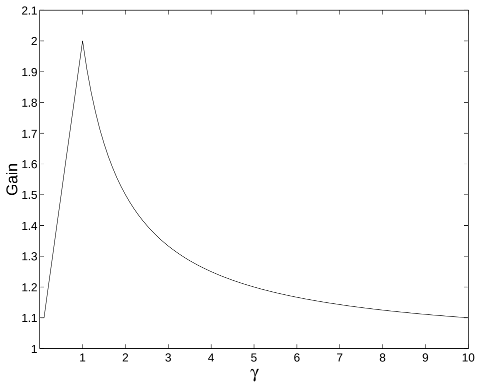


Figure 3: Gain of communication latency hiding.

We define a measure for the reduced runtime of algorithm $(A.2)$ versus $(A.1)$ as the *gain*

$$G = \frac{S_{lh}(p)}{S(p)} = \frac{1 + 1/\gamma}{\max(1, 1/\gamma)}.$$

Gain, G, can be interpreted as improved speedup due to latency hiding, and as improved efficiency: $G = S_{lh}(p)/S(p) = E_{lh}(p)/E(p)$, where $E(p) = S(p)/p$. Gain is always bounded by the constant 2, independent of the number of processors being employed. However, it should be pointed out that without a high value of gain (i.e., close to 2), utilizing larger numbers of processors will generally lead to low efficiency. Therefore, gain itself is a secondary objective which will lead to higher efficiency, which in turn leads to a solution that allows the use of a maximum number of processors. It is this point that we build upon as the main thesis of this work. If large numbers of workstations are to be used, one is forced into high-latency environments for economical or technical reasons. Thus, techniques are needed to exploit overlapping. Gain, as defined here, becomes a primary focus of the programmer and system designer, and therefore we now examine the relationships among gain, speedup, granularity, and communication overhead.

Figure 2 shows the speedup curves S and S_{lh} as a function of γ, and Figure 3 shows the corresponding gain. For $0 < \gamma \leq 1$, $G = 1 + \gamma$ is a linearly increasing function. This corresponds to case (c) in Fig. 1. The maximum gain $G = 2$ is obtained with $\gamma = 1$, where the maximum amount of communication latency hiding occurs. For $\gamma \geq 1$, $S_{lh}(p) = p$ is constant, and $G = 1 + 1/\gamma$ decreases asymptotically towards the limit 1. The communication overhead becomes negligible with increasing γ, and prevents any gain by latency hiding (cf. Fig. 1 (a)).

2.2 Extended Model

The goal of a refinement to the simple model is the incorporation of fundamental properties of the gain function as parameters.

Total calculation time is modeled as

$$t = \frac{i}{r_{op}},$$

where i denotes the number of instructions needed to sequentially solve the entire original problem with a given algorithm, and r_{op} the processor-specific processing rate in instructions per unit time. For the following, r_{op} is assumed to be constant.

Message transfer time can be described as follows:

$$t_{tr} = t_{cs} + \frac{b}{r_{com}} = \frac{i_{cs}}{r_{op}} + \frac{b}{r_{com}},$$

where $t_{cs} = i_{cs}/r_{op}$ is the overhead of the communication startup time, b the number of information units per message, and r_{com} the network transmission rate with respect to this unit of communication.

Communication time incorporates the number of messages transferred during a communication phase:

$$t_{com} = q\, t_{tr}.$$

The number of messages q depends on the algorithm, and often on the number of processors p. q may also depend on the hardware. It denotes the number of communications that are executed per task, assuming such communication operations are executed sequentially in each task. In case, the hardware supports multiple network interfaces, and messages are transferred simultaneously via these interfaces, this would be reflected in the value q by counting simultaneous message transfers as $q' = q/n_c$, where n_c is the number of parallel channels supported by the hardware.

Granularity is the parameter used to characterize both a parallel application and a parallel machine. The *software granularity* γ_s and *hardware granularity* γ_h are distinctly defined as

$$\gamma_s \stackrel{\text{def}}{=} \frac{f_i(p)}{f_b(p)}, \qquad \gamma_h \stackrel{\text{def}}{=} \frac{r_{op}}{r_{com}}.$$

The software granularity γ_s is the ratio of the number of instructions per task and the number of communication units, sent or received by each task. In this definition, f_i and f_b denote functions, which depend primarily on the number of processors p. Here, the simple case is studied, where the application is ideally parallelizable into p parts that are assumed to be identical in their calculation and communication requirements. In this case, software granularity characterizes the parallel program with $f_i(p) = i/p$ and $f_b(p) = b$:

$$\gamma_s = \frac{i/p}{b}.$$

The hardware granularity γ_h is the ratio of the instruction rate of each processor and the transfer rate of the network. In the following, the (collective) machine is assumed to be homogeneous, and the rates to be constant.

Restricting ourselves to the case of large messages, the speedup formulae become

$$\begin{aligned} S(p) &= \frac{p}{1 + q\gamma_h/\gamma_s}, \\ S_{lh}(p) &= \frac{p}{\max(1, q\gamma_h/\gamma_s)}. \end{aligned}$$

A particular latency hiding approach will introduce its own overhead, which depends on the hardware and the implementation of the message passing protocol. In principle, overhead increases both the calculation and communication parts of a program. However, to include the effect of this overhead, a simple term $\Omega \geq 0$ is added to the calculation part of the latency hiding version:

$$G(p) = \frac{1 + q\gamma_h/\gamma_s}{\max(1 + \Omega, q\gamma_h/\gamma_s)}. \qquad (3)$$

The term Ω models the qualitative overhead of communication latency hiding. Because accurate measures will be hard to obtain for quantitative modeling, we introduce a unit-less granularity here to examine the effect of varying this parameter.

3 Discussion of Latency Hiding Gain

Since we are interested in solving large scientific problems with large communication requirements, we direct our attention to the case with large messages. As given by equation (3), hardware granularity, software granularity and latency hiding overhead each appear as parameters.

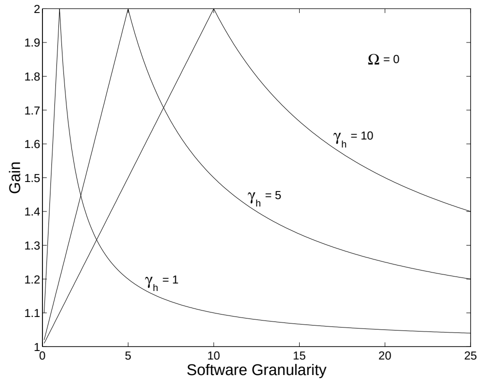


Figure 4: Gain dependency on granularity.

The dependency of gain on the granularities γ_h and γ_s is reciprocal. Figure 4 shows a family of gain curves versus software granularity for negligible overhead $\Omega = 0$. The curves are identical for gain versus hardware granularity. Due to the quotient γ_h/γ_s, points lying on the linearly increasing part of a curve in one case correspond to points on the decreasing part in the other case. For $\gamma_h, \gamma_s > 0$, G reaches its absolute maximum value of 2 for $\gamma_h = \gamma_s$. Two conclusions can be drawn from this figure:

(1) Larger granularities provide a certain degree of robustness concerning the range that provides a particular gain, both in a hardware and software sense.

(2) Smaller software granularity shifts the point of maximal gain towards smaller hardware granularity, and analogous, smaller hardware granularity shifts the point of maximal gain towards smaller software granularity. This result coincides with the intuitive need of faster communication networks for programs with smaller grain size.

Figure 5 illustrates the effect of overhead on the gain G. Four insights can be extracted from this figure:

(1) Increasing overhead results in decreasing maximum gain.

(2) Increasing overhead results in a smaller range of granularity, within which a particular gain can be obtained.

(3) To gain from latency hiding, the condition $\gamma_h/\gamma_s > \Omega$ must hold.

(4) The fact of most importance to the parallel programmer is the condition where the maximum gain can be obtained:

$$1 + \Omega = \gamma_h/\gamma_s.$$

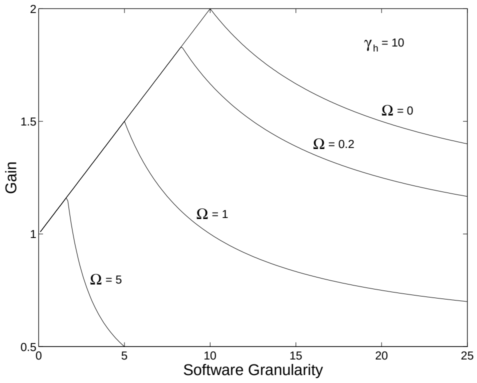


Figure 5: Gain dependency on overhead.

Under this condition, the maximum gain becomes

$$1 \quad < \quad \max(G) = 2 - \frac{\Omega}{1+\Omega} \quad \leq 2.$$

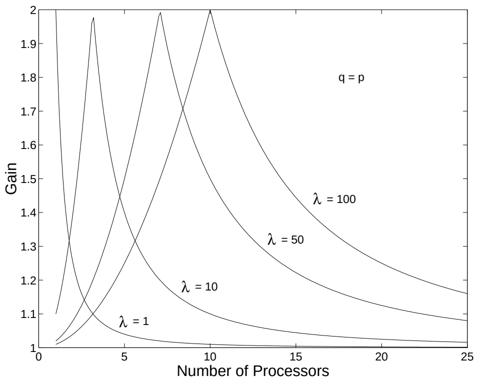


Figure 6: Gain dependency on number of processors.

To reveal the dependency of gain upon the number of processors, we make the number of processors explicit by introducing the *single-processor granularity* Γ_s:

$$\gamma_s = \frac{i/p}{b} = \Gamma_s/p.$$

Γ_s is the number of instructions of the sequentialized parallel program divided by the number of communication units each task communicates during the parallel execution. Combining software granularity and hardware granularity into the quotient $\lambda = \Gamma_s/\gamma_h$, the gain function rewrites to

$$G(p) \quad = \quad \frac{1 + pq/\lambda}{\max\Big(1+\Omega, pq/\lambda\Big)}. \qquad (4)$$

Figure 6 shows gain versus number of processors. Analogous to the two-processor system, the influence of overhead can lead to a loss (i.e. $G < 1$) if $\Omega > qp/\lambda$.

4 Experimental Results

In this section, a brief overview of some performance measurements are given that illustrate the validity of our model and the benefit of communication latency hiding with workstations in the Internet. We implemented communication latency hiding on workstations by refining the send and receive operations based on the TCP/IP protocol. For details see [7].

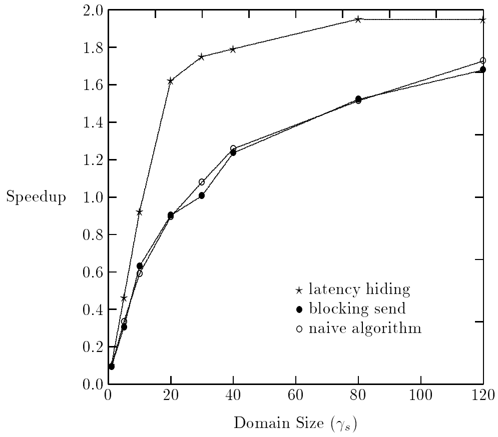


Figure 7: Speedup on the Internet.

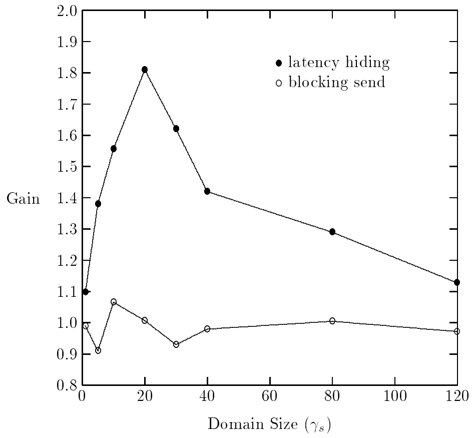


Figure 8: Gain on the Internet.

Recall the example of Section 2, and its parallel computation by means of algorithms $(A.1)$ and $(A.2)$. The gain of algorithm $(A.2)$ with respect to $(A.1)$ can be attributed to the socket buffer size. The UNIX send operation (system call `write`) blocks until the message has been passed to the transport layer (TCP). In our example, an explicit finite difference solver has been run on a rectangular domain with a fixed y-dimension of 1000 grid points. An artificial boundary condition yields 8000 bytes per message, using an 8 byte double precision number per grid point. With the default socket buffer size of 4 Kbyte, the send operation blocks, until the first 4 Kbyte part of the message is transferred via the network, and the remaining part of the message has been copied into the send buffer. Our latency hiding protocol utilizes the CPU cycles otherwise wasted during such send operations.

This mechanism accounts for the gain shown in Fig. 8. The measurements have been conducted with a two-processor configuration; one Sun SPARCstation at ETH Zürich, and one at MIT, Cambridge. The software granularity is varied by changing the mesh size of the x-direction of the domain. Figure 7 shows speedup and Fig. 8 gain versus a normalized domain size, i.e. software granularity. Normalization is with respect to a 10×1000 grid. Thus, a software granularity of 120 corresponds to a grid size of 1200×1000.

These measurements match the qualitative gain predictions of our communication latency model within the measurement accuracy of Internet communication. The overhead of our latency hiding implementation is small enough to deliver a maximum gain of more than 1.8. Furthermore, the data show that applications capable of overlapping communication with calculation, can perform on a high-latency network such as the Internet as efficiently as on a parallel machine, if $\gamma_s > \gamma_{s,max}$, where $\gamma_{s,max}$ is the software granularity that delivers maximal gain.

5 Conclusions

In this paper, we introduced the notion of latency hiding gain and a model that expresses the gain in terms of software granularity, hardware granularity and overhead. The model captures the opportunities and limitations of communication latency hiding by overlapping communication and calculation in a simple manner. We illustrated the validity of our model by presenting experimental data that also show the practical relevance of communication latency hiding for distributed parallel computing. For a more detailed treatment than possible here we refer the reader to [7].

Acknowledgements

We thank Walter Gander, Peter Arbenz, Christoph Sprenger and Edouard Bugnion for a fruitful environment. Thanks also to Charles Leiserson for his confidence to provide us with an account at MIT.

References

[1] P. Arbenz, C. Sprenger, H. P. Lüthi, and S. Vogel, "SCIDDLE: A Tool for Large Scale Distributed Computing," *Concurrency: Practice and Experience*. to appear.

[2] M. J. Atallah, C. L. Black, D. C. Marinescu, H. J. Siegel, and T. L. Casavant, "Model and Algorithms for Coscheduling Compute-Intensive Tasks on a Network of Workstations," *Journal of Parallel and Distributed Computing*, vol. 16, no. 4, pp. 319–327, 1992.

[3] V. S. Sunderam, "PVM: A framework for parallel distributed computing," *Concurrency: Practice and Experience*, vol. 2, no. 4, pp. 315–339, 1990.

[4] C. H. Cap and V. Strumpen, "Efficient Parallel Computing in Distributed Workstation Environments," *Parallel Computing*, vol. 19, pp. 1221–1234, Dec. 1993.

[5] "MPI: A Message-Passing Interface Standard." Message Passing Interface Forum, Apr. 1994. (Distribution: `netlib`).

[6] D. E. Comer and D. L. Stevens, *Internetworking with TCP/IP, Vol II: Design, Implementation, and Internals*. Englewood Cliffs: Prentice-Hall, 1991.

[7] V. Strumpen, "Communication Latency Hiding — Model and Implementation in High-Latency Computer Networks," Tech. Rep. 216, Department Informatik, ETH Zürich, June 1994. (Anonymous ftp: `ftp.inf.ethz.ch:/doc/tech-reports/1994/216.ps`).