Quantifying the Potential Benefit of Overlapping Communication and Computation in Large-Scale Scientific Applications

José Carlos Sancho Kevin J. Barker Darren J. Kerbyson
Kei Davis
Performance and Architecture Laboratory (PAL)
Computer and Computational Sciences Division (CCS-3)
Los Alamos National Laboratory, NM 87545, USA
jcsancho,kjbarker,djk,kei@lanl.gov

Abstract

The design and implementation of a high performance communication network are critical factors in determining the performance and cost-effectiveness of a large-scale computing system. The major issues center on the trade-off between the network cost and the impact of latency and bandwidth on application performance. One promising technique for extracting maximum application performance given limited network resources is based on *overlapping* computation with communication, which partially or entirely hides communication delays. While this approach is not new, there are few studies that quantify the potential benefit of such overlapping for large-scale production scientific codes. We address this with an empirical method combined with a network model to quantify the potential overlap in several codes and examine the possible performance benefit. Our results demonstrate, for the codes examined, that a high potential tolerance to network latency and bandwidth exists because of a high degree of potential overlap. Moreover, our results indicate that there is often no need to use fine-grained communication mechanisms to achieve this benefit, since the major source of potential overlap is found in *independent* work—computation on which pending messages does not depend. This allows for a potentially significant relaxation of network requirements without a consequent degradation of application performance.

Keywords: overlapping communication and computation, performance modeling, scientific applications

SC2006 November 2006, Tampa, Florida, USA

1 Introduction

The future of High Performance Computing (HPC) will inevitably be driven by the insatiable demand for ever-increasing computational power to meet the needs of larger and higher-fidelity simulations in scientific computing. Currently there are several proposed massively parallel machines intended to deliver petaflop (PFlop) or multi-PFlop performance, such as the machines being developed in DARPA High Productivity Computing Systems program [HPCS]. Despite being close to providing such capability, the key challenge is to construct such machines cost-effectively.

As the system size increases to tens of thousands of processors and beyond, interprocessor communication performance will become paramount in determining overall application performance. Therefore, system designers may be forced to develop and utilize increasingly fast networks to lessen communication delays. Unfortunately, such networks tend to disproportionately increase overall system cost.

To address this, an efficient approach may be to explore software mechanisms to achieve better system utilization and thus relax some requirements on the network. One promising software technique is the *overlapping* of communication and computation, or hiding communication delays using available computation subject to a data dependency analysis [Leu et al. 1987]. This technique has been explored mainly in the context of improving application performance by up to a factor of two when compared with a non-overlapping case. In contrast, our work here is to quantify an application's tolerance to lower network bandwidth and higher network latency. This could lead ultimately to a reduction in the cost of the network while having little impact on application performance.

Key issues addressed in this research are the quantification of the potential overlap that exists in scientific codes, the impact on application performance

and on network requirements, and the determination of whether fine-grained communication is important for exploiting such overlap. We develop a systematic method for identifying and quantifying potential overlap and use that information to quantify the sensitivity of application performance on network performance. Our technique is abstract with respect to programming language, communication layer, and the underlying hardware.

Figure 1 is a depiction of the temporal progression of a typical data-production, transmission, and consumption cycle. The essential point is that because data is aggregated for transmission, some data is potentially available before it is transmitted, and some is consumed only well after it is received. Depending on the order of production and consumption, and internal data dependencies, more finely-grained communication could yield potential overlap.

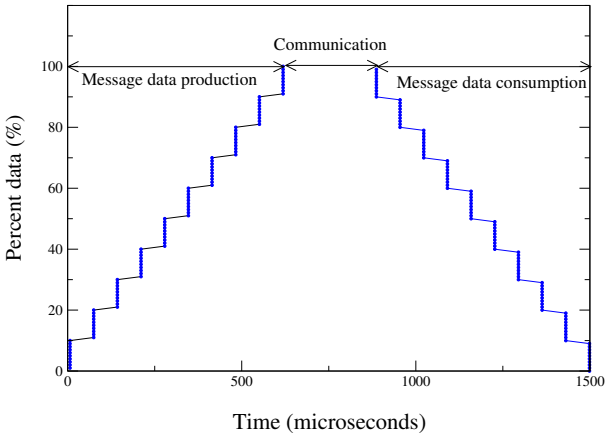

Figure 1: Percentage of Data Produced and Consumed Over Time on Data Transmitted in a Point-to-point Communication.

Our method is based on measuring the computation time available in the application to hide communication delays and then providing an estimate of the potential performance improvement given networks of varying performance characteristics. We illustrate the method on several large-scale production scientific applications using a 1,024-processor system. Results indicate that these scientific codes contain a substantial amount of unexploited overlap that could hide most communication delays.

The results indicate that the ability to hide communication delays is surprisingly independent of network latency and bandwidth. In addition, the major source of potential overlap comes from the availability of *independent* work—computation that does not generate data to be communicated—rather than *dependent* work in which the computation does effect the communicated data.

The rest of this paper is organized as follows. Section 2 reviews recent work dealing with the overlapping of computation and communication. Section 3 describes the experimental approach used in our analysis. Section 4 details the results obtained from this analysis for a number of large-scale applications. Section 5 concludes.

2 Related work

The benefit of overlapping computation and communication in parallel computing has been extensively studied in the past decade [Bell et al. 2006; Chen et al. 2005; Danalis et al. 2005; Iancu et al. 2005; Ke et al. 2005b; Quinn and Hatcher 1996; Sohn et al. 1996]. These studies empirically show that the benefit of overlapping is significant but strongly depends on several factors including the interconnection network and processor support, the parallel programming language, and the techniques used to extract the overlap.

Currently there is excellent support to exploit the potential overlap available in dependent work thanks to recent development in programming languages for high performance computing (such as UPC [Chen et al. 2005] and Co-Array Fortran [Coarfa et al. 2003]) and hardware support for addressing global data [Dunigan et al. 2005]. These provide light-weight one-sided communication, making fine-grained communication reasonably efficient.

Analysis of the efficiency of parallel programming languages such as UPC and communication libraries such as MPI in overlapping computation and communication has shown that UPC allows for substantial performance improvement when exploiting available dependent work using fine-grained message transfers for some benchmarks [Bell et al. 2006]. A similar study was performed using the communication library TACCS with similar conclusions [Danalis et al. 2005].

Two similar automatic runtime systems to exploit the overlap found in dependent work have been empirically evaluated [Iancu et al. 2005; Ke et al. 2005b]. Both approaches exploit the overlap at the receiver side by allowing the execution to proceed without waiting for the entire data transfer to be completed. Additionally, message pre-fetching techniques were explored in which processing nodes pre-fetch remote data before the remote node initiates a transfer [Liu and Abdelrahman 1998; Ke et al. 2005a]. Our approach is more general than these because it considers taking advantage of both independent and dependent work on both the sender and receiver sides.

A substantial departure from previous studies is that

we focus on characterizing the potential overlap from the application point of view without considering the underlying network and communication software implementations. We argue that this analysis is more useful for system designers because it facilitates prediction of potential application performance while exploring the network design space. In addition, previous work has concentrated on the benefit of overlapping communication and computation for standard benchmarks; our work complements those studies by evaluating the potential overlap for large-scale production codes.

3 Calculating Potential Overlap

In this section we describe our approach to quantifying potential overlap that can be used to hide communication delays. In general we are concerned with a single program multiple data (SPMD) programming model that corresponds to the execution model of most large-scale scientific applications. We consider two forms of SPMD processing, namely *concurrent* and *pipelined* [King et al. 1990]. In both forms, all nodes typically process their own sub-grids followed by boundary exchanges and possibly collective operations. The boundary exchanges are usually performed using point-to-point communication operations. In the concurrent case, all processors can progress through a single time-step without any inter-processor dependency, whereas in the pipelined case a processor can only progress after receiving data from processors located earlier in the pipeline.

To measure the amount of potential overlap it is important to know when a datum is *ready to transmit* at the sender and when a datum is *first-needed* at the receiver. We refer to the time at which data is ready to transmit as the *produce time* and the time at which it is first needed as the *consume time*.

Based on the produce and consume times we can calculate the potential overlap as shown in Figure 2 where the data produce time occurs at time *t0* and the possible consume times occur at *t1*, *t2*, and *t3*, where time progresses in the downward direction on the vertical axis. The time available for overlap is the difference between the produce and consume times. In the case that the consume time occurs at time *t1*, the datum is required at the receiver before the datum is produced by the sender and hence no overlap is possible, and similarly when the consume time *t2* is the same as the produce time *t0*. When the consume time occurs at later time *t3* the time available for overlap is *t3* - *t0*.

In this straightforward example of calculation of the potential overlap it is assumed that the processing nodes

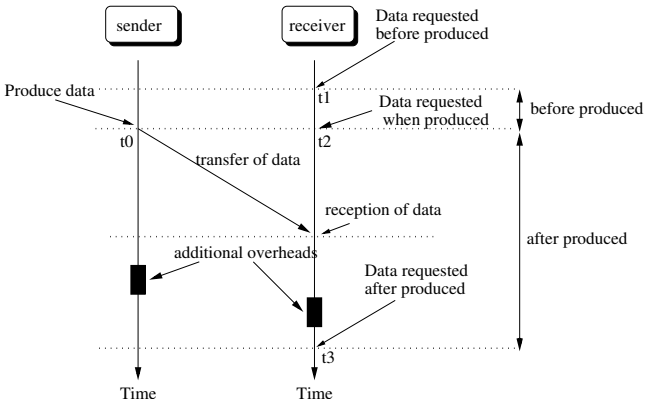

Figure 2: Calculation of the Potential Overlap in a Point-to-Point Communication.

are performing only computation between *t2* and *t3*. However, during this time the nodes may be involved in additional communication operations that could block execution, such as a collective or a blocking communication. In such cases the calculation of the potential overlap should not include the time spent in such communications. This is because communication operations result in the sharing of the network, and thus they can interfere with each other. The conservative approach that we follow is to subtract the time required for these communication operations when measuring the potential overlap. Note that some collective operations, such as reduction operations, perform computation on the local data before initiating the global reduction operation. For these operations the computation time component can be used in the calculation of potential overlap.

3.1 Communication and Computation Parameters

In order to compute potential overlap, applications are instrumented to collect the following information during their execution.

- **Amount of data produced and consumed (N_P and N_C):** the total data processed on a node in the produce and consume subroutines (measured in 8-byte words).
- **Amount of data sent and received (N_S and N_R):** the total amount of data exchanged when a processor exchanges boundaries with other processors in the system using point-to-point communication operations (measured in 8-byte words). This is obtained from the communication send/receive calls.
- **Produce and consume periods (T_P and T_C):** T_P is the average time interval between two consec-

utive data writes in the produce subroutine, and similarly T_C is the average interval between two data reads in the consume subroutine. These are measured by dividing the total elapsed time in the corresponding subroutine by the number of elements written or read for the data of interest.

- **Produce and consume order ($P(i)$ and $C(i)$):** The relationship between the order of data production and data consumption is required to accurately calculate the potential overlap. The function $P(i)$ defines the production order, and the function $C(i)$ defines the consumption order. In the case in which $P(i) = C(i)$ for all i, the data elements are produced in the same order in which they are consumed. The worst case in terms of overlapping is when $P(i) = C(N_C - i - 1)$, i.e., when the production order is the reverse of the consumption order. A complication to the definition of $P(i)$ and $C(i)$ is when temporary buffers are used, for example within multiple software layers in an application, which can alter the data ordering. In such cases, indirection functions, $index_{send}(i)$ and $index_{recv}(i)$, need to be known that effect the mapping between the produce and consumed data.

- **Additional independent computation time (T_{AP} and T_{AC}):** The average additional computational work occurring between the conclusion of the produce subroutine and the send communication call (T_{AP}), and the additional work between the receive communication call and the consume subroutine (T_{AC}).

To measure these parameters we use an application-level instrumentation library to provide accurate timing information as well as the other parameters on a per-processor basis. Instrumentation calls are manually inserted in the code. A cycle-accurate timer is used on the test system. The overhead of the timer in the application execution is 0.072 *ns* per measurement in our system as described later in Section 4.1. In order to minimize fluctuations due to cache misses and operating system effects, average times are measured over samples taken during execution and over all processors. Since in a parallel execution the amount of data processed may vary by processors, and thus the parameters N_S, N_R, N_P, and N_C may be different on each processor, in our analysis we show the values for the processors yielding the maximum value for each. This represents the worst case because the amount of data communicated is the largest, so that it will be more difficult to overlap with computation. Similarly in the case of applications where the amount of data processed per processor changes in the course of execution, such as in the application SAGE-AMR, we show the maximum value over the execution.

To illustrate the instrumentation, we consider part of a typical application consisting of several subroutines in which computation and communication take place within a loop as shown in Figure 3. In this example A is exchanged between processing nodes. A part of this code performs computational work on data that is not related to the data exchanged, referred to as *additional computational work*. The *produce subroutine* is the last subroutine where the data is written before it is sent. In the example *subroutine 3* is the produce subroutine. Similarly, the *consume subroutine* corresponds to where the data is first read after being received, *subroutine 0* in the example. Note that the produce and consume subroutines may be the same.

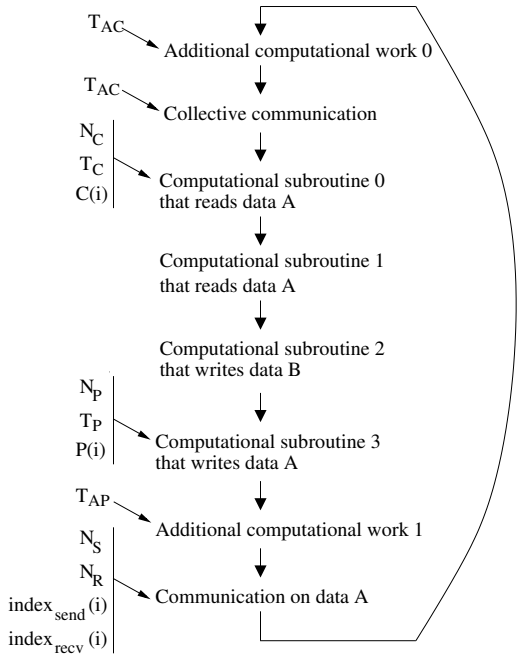

Figure 3: Example Program Structure Showing Required Measurements.

3.2 Concurrent Data Parallel Applications

In concurrent SPMD applications, processors perform the same operation at the same time but on different subgrids, and dependencies between processors only exist at the end of a computation stage. According to the model depicted in Figure 2, the concurrent model corresponds to the case of data being consumed after produced. This model occurs in large-scale applications such as POP [Kerbyson and Jones 2005; Jones et al. 2005], HYCOM [Halliwell 2004; Barker and Kerbyson 2005], and SAGE [Kerbyson et al. 2001].

The potential overlap is calculated using measurements collected from an application execution as described in

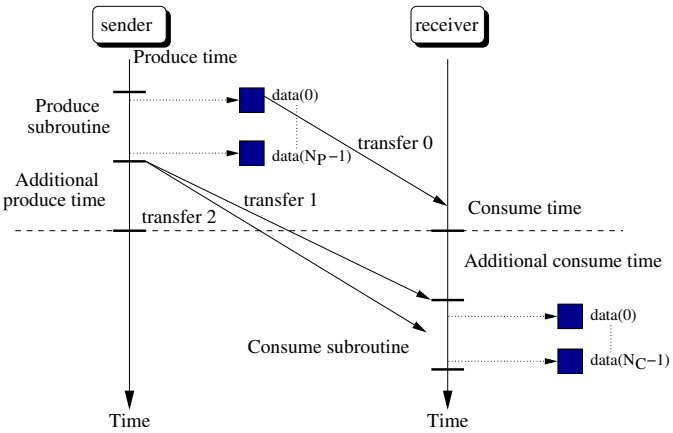

Figure 4: Example of Data Production and Consumption for Concurrent SPMD Applications.

Section 3.1. Figure 4 illustrates a single communication in the processing flow. The sender side consists of the produce subroutine and any additional produce time. In Figure 3, the additional produce time corresponds to *additional computational work 1*. Similarly, the receiver side consists of the consume subroutine and additional consume time. In Figure 3 the additional consume time corresponds to the *additional computational work 0*. The dotted horizontal line in Figure 4 represents a point-to-point communication. The location of this serves as a reference point separating the sender and receiver components. The time available for hiding the communication of data element i is given by

$$t(i) = T_{AP} + T_{AC} + T_P * (N_P - P(i) - 1) + T_C * C(i)$$

The component $T_P*(N_P-P(i)-1)+T_C*C(i)$ represents the time available for hiding the communication within the dependent work, and $T_{AP} + T_{AC}$ corresponds to hiding the communication delay using the independent work. Note that the time available from the dependent work is heavily dependent on the produce and consume index orders.

An important consideration is the granularity required for the message transfers. Three basic cases are considered. First, the data is sent during the produce subroutine as shown in Figure 4 as *transfer 0*. In this case, data are sent as soon as they are produced. This case exploits the dependent work of the produce subroutine ($T_P * (N_P - P(i) - 1)$) to hide communication delays. A fine-grained message transfer is needed to minimize the overheads of multiple message transfers. Network latency and bandwidth should not be limiting factors of performance, due to the small message sizes and pipelined message transfers. However, the message injection rate (the g parameter of the network model described in Section 4.2) and the overheads of sending and receiving multiple messages may make this approach to exploiting overlap inefficient.

The second case, *transfer 1* in Figure 4, corresponds to sending the data after it is completely produced but the data arrives at the receiver before the consume subroutine. This case exploits the independent work available ($T_{AP} + T_{AC}$). Unlike the previous case, there is no need to use fine-grained message transfers because all necessary data has been produced at the time of initiating the data transfer. We can take advantage of the communication optimization based on aggregating data into large messages to minimize communication overheads. Hiding communication delays with independent work is important because the network injection rate will not be a limiting factor and the aggregate communication overheads are not as high as with many smaller messages. Thus this approach relaxes the requirements placed on the communication subsystem, allowing for a low network injection rate and higher communication overheads. It is also likely that more independent work than dependent work will be available.

Finally, the last case considered is similar to the previous one, but the data is received during the consume subroutine. Here both independent and dependent work of the consume subroutine are exploited ($T_{AP} + T_{AC} + T_C * C(i)$). As before, there is no need to use fine-grained message transfer because all data has been produced before initiating the transfer. Nevertheless, to exploit the dependent work in the consume subroutine, a software or hardware mechanism may be required to identify when the data is received [Iancu et al. 2005].

3.3 Pipelined Data Parallel Applications

In pipelined SPMD applications, processing nodes carry out the same calculations on their data sub-grids but with the added requirement that there is a dependency in the processing flow. Processing nodes operate in a pipeline: they process their sub-grid after receiving data from nodes located earlier in the pipeline and send results to nodes further down it. This mode is exemplified by *wavefront* applications such as Sweep3D [Hoisie et al. 2000; Koch et al. 1992].

Since the receiving nodes wait on data for immediate use, the model corresponds to the case of data being consumed before it is produced (Figure 2). Therefore, the resulting time available for overlap is zero while the pipeline is being filled. Figure 5 illustrates the calculation of the potential overlap for these codes. The receiver is blocked at the consume subroutine waiting for data from the sender. Meanwhile, the sender is executing the produce subroutine. In that model there is no additional produce and consume times because the synchronization for the completion of the data reception is performed immediately after the data is produced at

the sender. In the case of concurrent SPMD applications this synchronization is performed later since both the sender and receiver perform the same work at the same time.

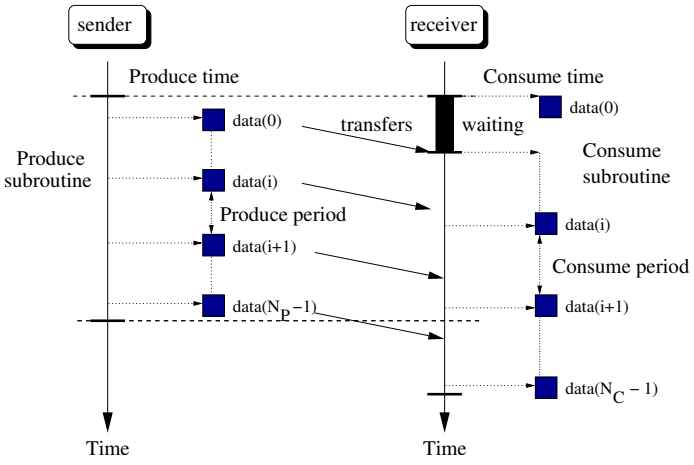

Figure 5: Example of Data Production and Consumption for Pipelined SPMD Applications.

The only potential overlap is the dependent work in the consume subroutines. We have seen that there is no time available for the first needed data, but the next data can use the computational work on the previously received data to hide the communication delays. A fine-grained message transfer is required to exploit the dependent work in the produce subroutine because data needs to be sent as soon as it is produced. The efficiency of this approach strongly depends on several factors. First is the ordering of the data between the produce and consume routines. Recalling the previous section, the worst scenario—where the first consumed datum is last produced—will nullify the ability to overlap communication with computation for these applications. Second, the efficiency depends on the injection rate of the network and the produce period (T_P) of the data in order to evaluate whether or not the data is injected into the network at least as fast as the receiver needs it. Otherwise, the receiver will be blocked waiting for data to be received. In order to analyze this behavior the ratio $\frac{T_C}{max(T_P,g)}$ should be evaluated.

3.4 Techniques to Maximize the Potential Overlap

The amount of independent work exhibited by the application can often be increased through code modification in order to better hide communication delays. The techniques explored in this paper are briefly described below.

- **Subroutine re-arrangement:** Computational routines may be moved in the code to provide additional produce and consume time. For example, *subroutine 2* may be moved after *subroutine 3* in Figure 3, increasing the additional produce time. A data dependency analysis is required in order to ensure correctness.
- **Loop indexing:** In domain decomposition applications, the domain is usually decomposed into inner and outer regions [Hiranandani et al. 1992]. Boundary data to be communicated may be produced earlier by updating the outer regions first followed by the inner regions. This will increase the additional produce time as well as the additional consume time. This technique requires rearrangement of the loop indexing in the produce and consume computation subroutines. A data dependency analysis is again required in order to ensure correctness.
- **Loop distribution:** This technique separates independent computation from dependent computation in a single loop into multiple loops [Hiranandani et al. 1992]. The result can be to increase the additional produce and consume times.

In this analyzes we provide an estimate of the amount of independent work available if these techniques were to be implemented in the codes. We estimate the independent work using *subroutine re-arrangement* by measuring the computation time within each subroutine. In the case of the *loop indexing* technique, the additional produce time available is estimated based on the produce period and the interior data that will not be communicated, $T_P * N_{inner}$, while the additional consume time is estimated based on the consume period, $T_C * N_{inner}$. Finally, in the case of *loop distribution*, we measure the time of a loop that only performs the independent calculations in the application.

In the applications evaluated in this paper only POP and SAGE implement techniques to overlap communication with computation. MPI non-blocking send and receive operations are used to exploit the independent work such as performing local copies and initializing ghost cells. In this paper, we explore other sources of overlap related to the computational core of the application rather than just communication-related operations since communication alone may not provide enough overlap or may differ in other systems.

4 Experimental Results

We begin with a description of the parallel system that served as the experimental platform. Next we describe

Table 1: Application Input Decks

Application	Input	Global domain (cells)	Scaling mode	Number of processors
HYCOM	large.inp	$4,500 \times 3,298 \times 26$	Strong	1,006
POP	x1	$320 \times 384 \times 40$	Strong	128
SAGE	timing_h	36 million	Weak	1,024
SAGE-AMR	timing_b	52-74 million	Weak	1,024
Sweep3D	mk=1	$320 \times 320 \times 400$	Strong	1,024

the model used to estimate the communication delays when transmitting data through a given network. We then describe several large-scale parallel applications. For each application we analyze the potential overlap and the sources from which overlap may be obtained. Finally we evaluate the impact of overlapping on application performance as a function of network performance.

4.1 Experimental Set-up

The parallel computer used in the evaluation is representative of current high-performance computing platforms, with a peak performance of 4 TFlops. It is composed of 256 dual-processor dual-core AMD Opteron nodes, for a total of 1,024 processing cores. Each node runs the Linux operating system. The nodes are connected using a Voltaire 288-port InfiniBand 4x switch. The processor's clock speed is 2 GHz. Each core has a 1 MB L2 cache, and each node has 4 GB of physical memory.

A range of network performance characteristics are considered in order to analyze the influence of network performance on the potential overlap. Bandwidths ranging from 1 MB/s up to 5 GB/s, and latencies of 1 µs, 2 µs, 4 µs, and 8 µs are considered. Though most of today's interconnects do not provide the higher-end performance characteristics (bandwidth of 5 GB/s and latency of 1 µs), such performance may be more common in the near future, motivating the need for modeling communication delays rather than measuring them in real systems when evaluating the potential overlap. As a reference point for today's network performance, we considered the performance of the MVAPICH [MVA] message passing system on InfiniBand SDR 4x that delivers a network latency for small messages of 4 µs, and a uni-directional bandwidth of 950 MB/s for large messages.

4.2 Estimating the Communication Delays

To model the time when transferring the data through the network using point-to-point communication operations we use the communication model described by [Bell et al. 2003]. This model captures the behavior of non-blocking communication operations in current interconnection networks. In this model, sending a message between two processors is approximated by $EEL + G \times size$, where *size* is the message size in bytes, *EEL* (end-to-end latency) is the the total time for a message from the beginning of a send until receipt, and G is the additional gap per byte in large messages. The reciprocal of G is then effective maximum bandwidth of a network channel. A parameter g is defined as the elapsed time between two consecutive small messages—the inverse of the message injection rate.

The resulting value for *EEL* is actually slighter lower than the typical measured *EEL* from a ping-pong evaluation benchmark that includes the send and receive overheads and the transport of the message, because they are likely to overlap each other in current interconnection networks. We will use the terms *EEL* and network latency interchangeably. In this model we are assuming that there is no contention in the network, where multiple messages are competing for the same network channel.

Collective communication operations are modeled using a binary "fan-in, fan-out" message passing strategy in a binary tree structure which takes $2 \times log_2(N) \times EEL$ for small messages (typically the case for most of the scientific applications), where N is the number of processors.

It is beyond the scope of this paper to analyze how the potential overlap can be fully extracted from the application using specific communication systems. We assume that this overlap can be fully extracted as evidenced in recent studies [Lawry et al. 2002]. Basically, the ability to exploit the potential overlap present in the application is influenced by the capability of the network hardware and message passing system to perform the transfer concurrently while the processors compute. Some commodity interconnection networks, such as InfiniBand, already provide specialized hardware support, such as DMA engines, and MMU's (full memory management units) in the network interface cards. These networks can perform a remote transfer from a source application virtual address to a destination virtual ad-

Table 2: Measurements on Dependent Work

	HYCOM			POP		SAGE	SAGE-AMR	Sweep3D	
Parameter	*ubavg*	*pbavg*	*vbavg*	*R*	*Q*	*vctrp*	*vctrp*	*phiib*	*phijb*
T_P (*ns*)	28	32	28	11.6	10	5.5	7.9	1,050	18.4
T_C (*ns*)	43	43	43	9	2.5	22.4	24	1,050	45
N_C (*8-byte words*)	12,100	12,100	12,100	1,248	1,248	36,424	127,104	10	10
N_P (*8-byte words*)	11,236	12,100	11,664	1,248	1,248	35,096	71,952	10	10
N_S, N_R (*8-byte words*)	11,232	11,232	11,232	352	352	36,424	127,104	10	10

Table 3: Measurements on Independent Work (time in µs)

	HYCOM			POP		SAGE	SAGE-AMR	Sweep3D	
	ubavg	*pbavg*	*vbavg*	*R*	*Q*	*vctrp*	*vctrp*	*phiib*	*phijb*
T_{AP}	0	576	282	2.15	2.13	0	0	0	0
T_{AC}	0	0	0	0	3.36	0	0	0	0
Loop distribution	192	192	0	0	0	0	0	0	0
Produce loop indexing	0	27.7	12.2	10.3	8.9	0	0	0	0
Consume loop indexing	37.3	37.3	37.3	8	2.2	0	0	0	0
Re-arranging subroutines	0	0	0	0	0	1,943	3,061	0	0
Communications	0	0	0	2.8	2.8	0	0	0	0
Total	229.3	833	331.5	23.25	19.39	1,943	3,061	0	0

dress without processor intervention, substantially reducing the CPU overhead of sending messages. Low communication overhead is important, especially in the case of exploiting the dependent work in the produce subroutine, otherwise it negatively offsets the advantages of overlapping.

The ability to overlap is also dependent on the message passing system. MPI provides non-blocking send/receive operations and the MPI-2 additionally provides one-sided communication operations, which allow the overlap of communication and computation. Actual realization strongly depends on the implementation of the message passing system: whether implementation makes full use of the network hardware, and whether communications can progress independently of further calls to the communication library.

4.3 Overlap Analysis

We examine four large-scale parallel applications. HYCOM and the Parallel Ocean Program (POP) are both ocean modeling codes that represent water regions as 3D regular grids. However, these codes differ in the methods in which they model fluid flow and mixing in regions of varying depth. SAGE is a hydrodynamics code to simulate shock-wave propagation with the capability of using adaptive mesh refinement (AMR), where unlike the other applications examined the amount of data processed per processor may change during the execution. Finally, Sweep3D is a kernel application which implements the main processing involved in S_N particle transport. The input decks for each application are summarized in Table 1. For each of these applications we collected the parameters described in Section 3.1, and the independent computation times that could be extracted if the techniques described in Section 3.4 were implemented, to give a total potential overlap. Tables 2 and 3 contain the parameters related to the dependent and independent work, respectively.

We normalized the potential overlap ($T_{overlap}$) to the communication delays of the point-to-point operations (T_{comm_point}), $\frac{T overlap}{T comm\_point}$, in order to quantify how much potential overlap is available relative to their communication delays. A normalized overlap greater than or equal to 1 means that the point-to-point related communication delays can be fully overlapped. The normalized overlap is useful for explore the robustness of the potential overlap to variations in the system such as increasing the processor performance, or the existence of contention in the network. The influence of the processor performance can be viewed as decreasing the potential overlap latent in the application. For example, a normalized overlap of 2 means that the potential overlap is still large enough to support a processor of twice performance without incurring communication delays.

4.3.1 HYCOM

In HYCOM the *barotropic* computational routine models vertical fluid mixing and is the most time-consuming.

The boundaries exchanged consists of three data structures (*pbavg*, *ubavg*, and *vbavg*) at the beginning of each iteration.

An interesting observation is that the amount of data produced for the data structures *pbavg* and *vbavg* is larger than the amount of data actually transmitted, see Table 2. We can take advantage of this difference by applying the *loop indexing* technique to first produce the data that will be communicated, and then produce the remaining data. This additional time can be used to overlap communication with computation. Table 3 shows that this elapsed time is 27.7 µs for *pbavg* and 12.2 µs for *vbavg*. Similarly, the amount of data consumed for the data structures is larger than the amount of data actually transmitted. The resulting additional time obtained using the *loop indexing* technique to first consume the data that is not being communicated is 37.3 µs for each of the data structures analyzed.

All three data structures are communicated simultaneously, thus there is potentially significant elapsed time between the production of the first data and its transmission. Similarly, there is potentially time available, although less, between production of the second data and its transmission. The resulting additional time on the producer side is shown in Table 3. Note that no time is available between production of the last data structure and transmission; however 576 µs and 282 µs are available between production and transmission (T_{AP}) of *pbavg* and *vbavg*, respectively.

Since no additional time is available, the communication associated with *ubavg* is the most difficult to overlap with computation. However, on the consumer side it is possible to use the *loop distribution* technique to obtain some available computation time. In this case, it is possible to hoist an independent calculation ahead of the message reception operation, providing 192 µs of computation for communication overlap. Note that while this is useful for hiding communication delays associated with *ubavg* and *pbavg*, it is not applicable to *vbavg* because of a data dependency. Note that this analysis is optimistic because we are assuming that there is no overhead involved when implementing this optimization technique in the code.

The total potential overlap is given in Table 3. For *ubavg*, *pbavg*, and *vbavg*, these times are 229.3 µs, 833 µs, and 331.5 µs, respectively. Figure 6 shows the normalized overlap for different bandwidths and latencies in a generic network. As can be seen, the normalized overlap linearly increases as the network performance improves, reaching 12 in the highest network performance (bandwidth of 5 GB/s and latency of 1 µs). This is an expected behavior because the resulting communication delays are becoming smaller when the network performance improves while the potential overlap remains the same. Even in the experiment network (InfiniBand) the normalized execution time achievable is still significant, 2.32.

4.3.2 POP

POP spends most of its time in a preconditioned conjugate gradient solver in the barotropic method which iterates computing and communicating until a solution is reached or 1,000 iterations have been performed. Several preconditioned conjugate-gradient solvers are implemented in the code, but the results presented in this paper correspond to the *PCG* solver. In each iteration of this solver POP exchanges its boundaries in two data structures named R and Q. The boundary exchange is performed using two independent point-to-point communication calls that are scattered throughout the code. A reduction collective communication operation named *global_sum* is performed immediately after each of these communications using a summation operation. Inside this function POP computes a local sum on the data prior to calling to the *MPI_Allreduce* that performs the global summation operation among the nodes. This summation on the local data can be considered an additional consume time that does not involve computation on the communicated data, R and Q, and is performed after the point-to-point communication operation but before the communicated data is first requested. The time to perform this summation is 2.8 µs (shown in Table 3).

There is some additional independent computation time after the corresponding produce subroutine (T_{AP}) for the data R and Q, 2.15 µs and 2.13 µs, respectively. There is also an independent computation after the point-to-point communication operation on the data Q before that data is requested in the consume subroutine (T_{AC}), providing 3.36 µs.

Moreover, because the number of data elements transmitted (352) is less than the number produced (1,248), there is the possibility of extracting some overlap with independent computation via *loop indexing*. The potential overlap is 10.3 µs and 8.9 µs for R and Q, respectively. Similarly, the amount of data consumed is greater than that communicated, so there is some additional independent time that can be extracted via *loop indexing*. A total of 8 µs and 2.2 µs can be extracted for R and Q.

As can be seen from Table 3, the total potential overlap for both data elements is 23.25 µs and 19.39 µs, respectively. These times are large enough to hide the communication delays of the point-to-point communication operations. As can be seen in Figure 6, there

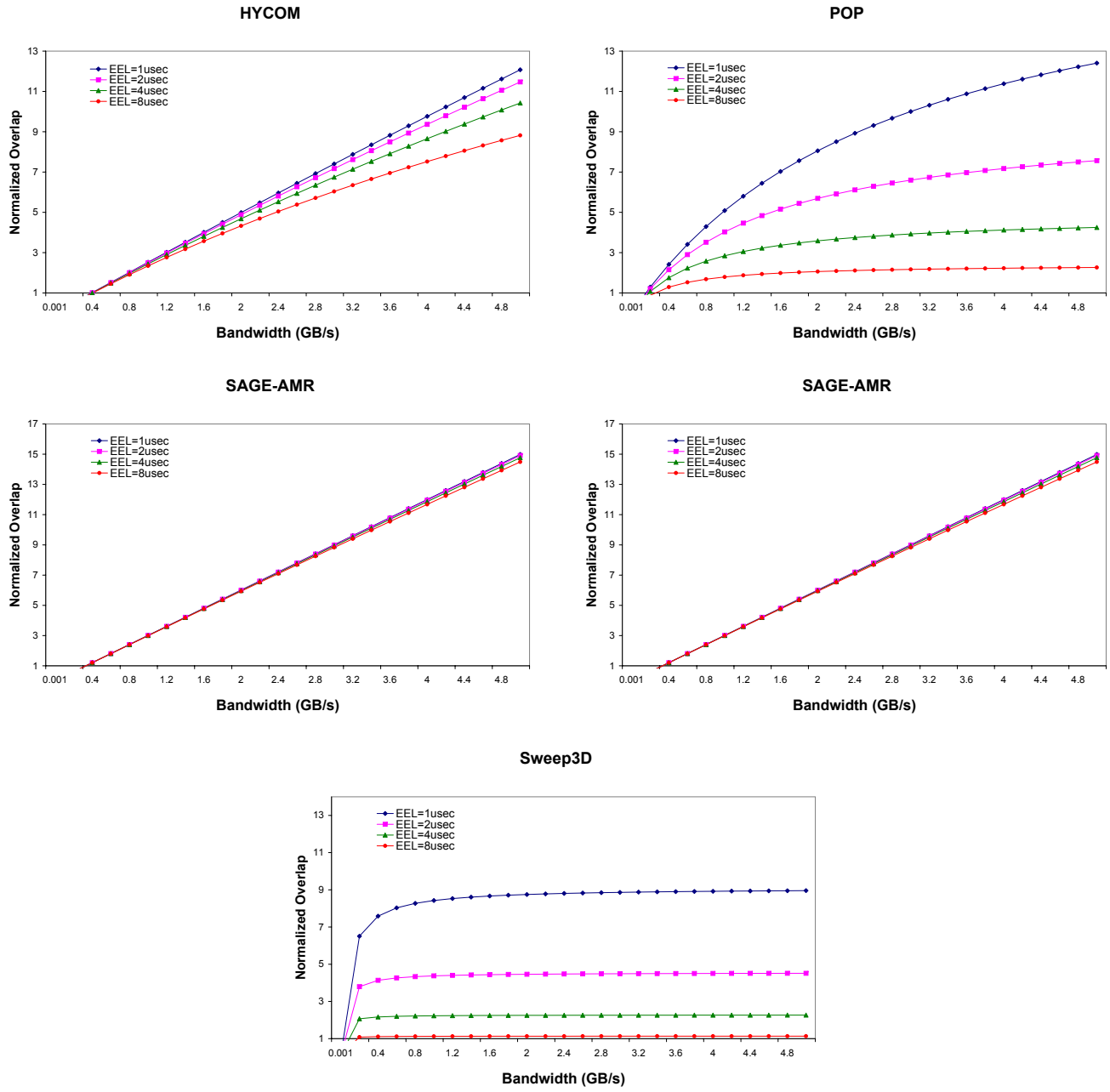

Figure 6: Normalized Overlap for Each Application.

is a normalized overlap of 12.4 at the high end of the considered network performance, while for the testbed the normalized overlap of 2.78 is achievable.

An interesting observation for POP is that the normalized overlap is very sensitive to the latency of the network. The normalized overlap increases from 2.2 to 12.40 when the network latency is reduced from 8 µs to 1 µs (in the case a network bandwidth of 5 GB/s). This is because with the data input *x1*, running on 128 processors, the communication delays are mainly dominated by the latency rather than the bandwidth.

4.3.3 SAGE

SAGE, like POP, is heavily dependent on the performance of a conjugate-gradient solver subroutine, which in this case is the *CG* solver. In it SAGE performs exchanges using a point-to-point communication operation on a data structure named *vctrp*. Unfortunately, because this data is produced immediately before it is transmitted, and is consumed immediately after it is received, there is no additional independent computation to hide the communication delays associated with the transfer of the data. However, some independent computation can be extracted by rearranged subroutines. By moving an independent subroutine typically executed before the point-to-point communication operation to after the communication call, 1,943 µs and 3,061 µs are available for overlap in SAGE and SAGE-AMR, respectively. The communication delays for SAGE are high because of the large amount of data transmitted. However, the normalized overlap achievable is large as shown in Figure 6. For a high performance network the normalized overlap is 34.04 and 15.58, for SAGE and SAGE-AMR, and in the testbed the normalized overlaps are still significant, 6.25 and 2.84, respectively.

4.3.4 Sweep3D

Sweep3D, unlike the previously examined applications, is a pipelined SPMD application, meaning that communication and computation progresses through the multi-dimensional processor array in a wavefront pattern.

Two data structures called *phiib* and *phijb* are communicated between processors using point-to-point communication operations, and two primary loops, one to update each data structure, provide the bulk of the computation. For these applications there is no independent work available to overlap with this computation time.

The only computation available for overlap is therefore related to work dependent on the data *phiib* and *phijb*. The produce/consume period for *phiib* is 1,050

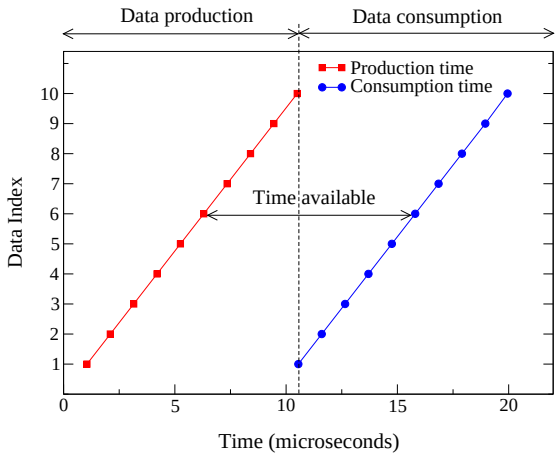

Figure 7: Potential Overlap Time for Each Datum in *phiib*.

ns which is much higher than the one corresponding to *phijb* which is 45 ns. For *phijb* it is important to check the ordering of the produced and consumes data. Fortunately this order corresponds to the best scenario, $p(i) = c(i)$ for all i. However, the value of $\frac{P_C}{max(P_P,g)}$ will be dominated by the parameter g since in today's interconnection networks this parameter is unfortunately larger (1 µs) than the produce period, 45 ns, eliminating the possibility of overlapping of *phijb*.

For *phiib* the key factor to exploiting the dependent work is also in the ordering of the data. Since this data is not needed until all data in *phijb* are finally produced the model that follows the data *phiib* to overlap is much like the one for concurrent SPMD applications with the exception that there are no additional produce and consume independent times (T_{AP} and T_{AC}). The dependent work is defined by $T_P * (N_P - P(i) - 1) + T_C * C(i)$. Figure 7 shows the resulting potential overlap calculated for each data based on this formula. As can be seen, there is a uniform computation time of 9.46 µs available to hide the communication delays across all data. This time yields a normalized overlap of 8.96 (Figure 6) in the high-end network performance scenario, while for the testbed it is 2.31.

Note that although the dependent work can also be exploited for the other applications it is more efficient to exploit only the independent work because the requirements to exploit the independent work to hide communication delays are simpler, and for those applications independent work is sufficient to hide the communication.

4.4 Impact on Application Performance

To evaluate the impact of overlapping on performance we normalize the total execution time to the time for computation only, $\frac{Tcomp+Tcomm}{Tcomp}$, where T_{comp} is the computation time and T_{comm} is the communication time corresponding to both point-to-point messages and collectives. This metric provides an indication of actual execution time relative to an idealized execution time in which effective communication delays are zero (full communication overlap). Table 4 summarizes the computation time for each application using the number of processors specified in Table 1. Note that this computation time is not the total execution time of the application, rather it is the time for a single iteration of the part of the application code previously discussed. The total running time of the applications will also depend on the execution time of the other parts of the code.

Table 4: Computation Time

Application	Time (µs)
HYCOM	2,588
POP	31.46
SAGE	5,577
SAGE-AMR	9,170
Sweep3D	10.08

Figure 8 shows the normalized overlapping and non-overlapping execution times for the applications considering a network latency of 4 µs(current InfiniBand latency). As can be seen for all the applications overlapping converges faster than the non-overlapping case as the network bandwidth increases. In the particular case of HYCOM, overlapping can achieve the ideal scenario of zero effective communication delays, even at very low bandwidth (400MB/s), while the non-overlapping case asymptotically converges only at much higher bandwidths. The significance of this result is that one of the major benefits that overlapping provides is a reduction in the requirements in the network performance. Note that POP and SAGE are not reaching the ideal case even when the point-to-point communications are fully overlapped because the collective communication operations are not regarded as being possible to overlap in this treatment. In the case of Sweep3D, the ideal case is not reached because the point-to-point communication on the data *phijb* also cannot be overlapped, as was previously shown

For the purpose of illustrating the bandwidth requirements of the application when overlapping, Figure 9 shows the bandwidth requirements at the point that overlapping achieves the same performance as non overlapping with a network bandwidth of 5 GB/s. The network bandwidth is an order of magnitude lower than that required when not overlapping for the network latencies evaluated. These results demonstrate that overlapping becomes an important optimization technique to relax the requirements in the network bandwidth. The bandwidth requirements are decreasing when the latency decreases, as can be clearly seen for Sweep3D, because the bandwidth is more predominant in the communication delays at lower network latencies, and thus when overlapping we need more bandwidth to achieve the same performance than when not overlapping.

Also, for the network latency requirements we can see a significant relaxation when overlapping with respect to the case of non overlapping. Table 5 shows the normalized execution time when varying the network latency assuming a fixed network bandwidth of 5 GB/s. As is shown, the same performance can be achieved with higher latency than when not overlapping. In particular, for Sweep3D the normalized execution time when overlapping at a latency of 8 µs is the same (1.79) as when not overlapping at 4 µs. However, for other applications such as POP, overlapping cannot achieve the same performance as achieved when reducing the network latency even when the point-to-point communication operations are fully overlapped. The reason is that for this particular application the communication delays are mostly dominated by the collective communication operations as shown in Figure 10, which shows the fraction of the time that the applications spend on computation, point-to-point communication, and collective communication assuming the same network configuration as the one considered in Figure 9. For these applications reducing the network latency has more impact on performance than overlapping communication with computation. Mechanisms to overlap the collectives as well as the point-to-point communication operations are needed in order to substantially relax the requirements on the network latency.

In addition, we can see for these applications that overlapping becomes crucial to performance because simply increasing the network bandwidth is insufficient to increase performance. Applications that are latency sensitive rather than bandwidth sensitive are the ones that can substantially benefit from overlapping. For example, for Sweep3D, the normalized performance achieved when not overlapping at a high network bandwidth (5 GB/s) is 1.2 (see Table 5) while overlapping can further improve it to 1.1 at a significantly lower network bandwidth. This result is important because of physical lower bounds on network latency in large-scale computers because of the distance that signals must travel between processors. Then overlapping becomes a crucial optimization to improve the application performance rather being used as an optimization to relax the network requirements.

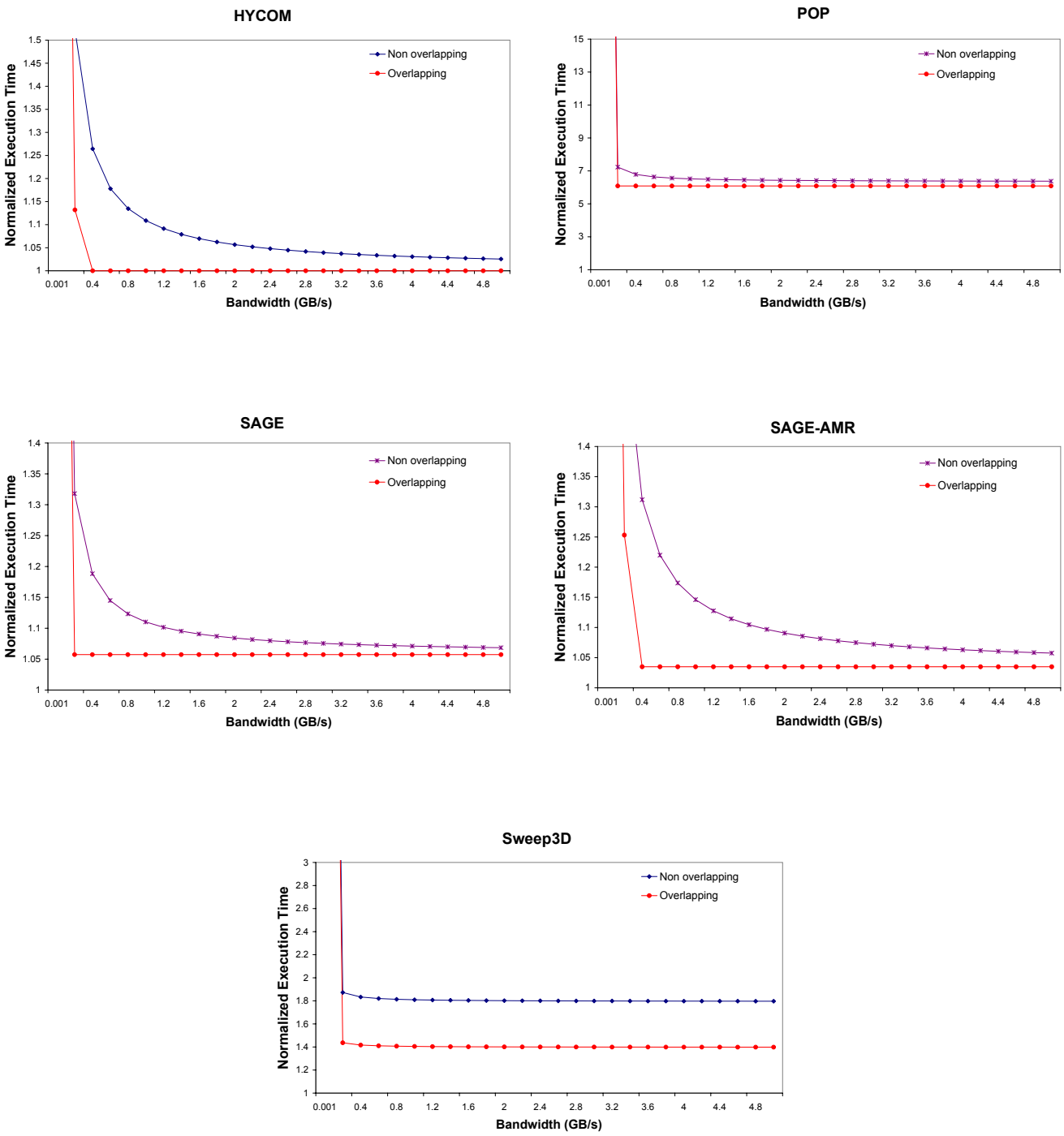

Figure 8: Normalized Execution Time for Each Application.

Table 5: Normalized Execution Time for Non Overlapping and Overlapping for Different Network Latencies

	HYCOM		POP		SAGE		SAGE-AMR		Sweep3D	
Latency (µs)	Non over.	Over.	Non over.	Over.	Non over.	Over.	Non over.	Over.	Non over.	Over.
1	1.02	1.00	2.37	2.27	1.02	1.01	1.03	1.00	1.20	1.10
2	1.02	1.00	3.70	3.54	1.03	1.02	1.03	1.01	1.39	1.20
4	1.02	1.00	6.37	6.08	1.06	1.05	1.05	1.03	1.79	1.39
8	1.03	1.00	11.71	11.07	1.12	1.11	1.09	1.06	2.59	1.79

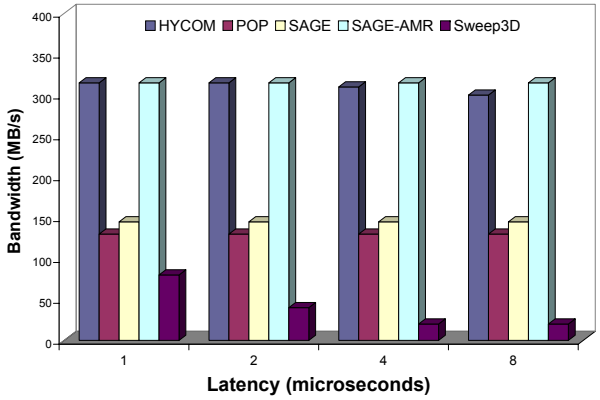

Figure 9: Bandwidth Requirements for Each Application when Using Overlapping that Achieves the Same Performance as Non-Overlapping with a Network of 5 GB/s.

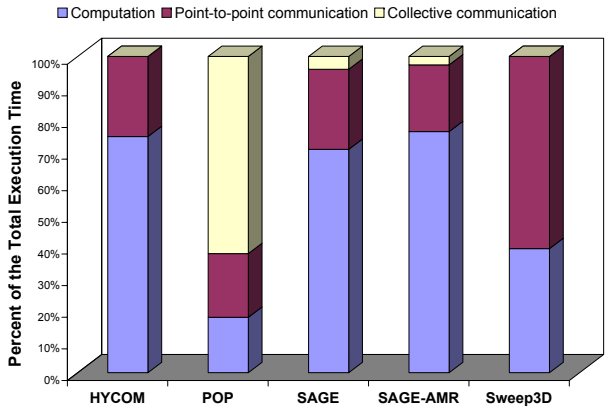

Figure 10: Percent of the Total Execution of the Time Spend in Performing Computation, Point-to-point and Collectives Communication Operations.

5 Conclusions

We have analyzed the potential overlap between communication and computation in several current large-scale production applications. A method has been developed to evaluate the potential performance improvement for those codes that does not require rewriting the applications for a specific communication software library or parallel programming language. The main advantage is that it simplifies evaluating the benefit of overlapping communication and computation for networks with various performance profiles.

An analysis of several large-scale applications executed at a scale of 128 to 1,024 processors has shown that they latently highly tolerant to network latency and bandwidth. This is because the potential available overlap is large enough to hide the communication delays. The time available for overlap is often as much as twice what is required to effectively hide the communication delays with current network technology. The significance of this result is that the potential performance of these codes is relatively independent of network performance, allowing for a relaxation of network requirements. Moreover, the major source of overlap comes from the *independent* work, eliminating the need for fine-grained communication mechanisms.

This result indicates that for a potentially large class of real-world applications, future high performance computing systems could use lower performance and more cost-effective networks without negatively impacting application performance if the overlap of communication with computation were fully exploited. In addition, overlapping can lead to improvements in application performance on existing platforms. Overlapping is critical in this regard, as these results also indicate that increases in network bandwidth alone may not be sufficient to increase workload performance.

Acknowledgments

This work was funded in part by the Accelerated Strategic Computing program of the Department of Energy, and in part by the Defense Advanced Research Projects Agency (DARPA) through the High Productivity Computing Systems program, Los Alamos is operated by the Los Alamos National Security, LLC for the US Department of Energy under contract No. DE-AC52-06NA25396.

References

Alexandrov, A., Ionescu, M. F., Schauser, K. E., and Scheiman, C. 1995. LogGP: Incorporating long messages into the LogP model - One step closer towards a realistic model for parallel computation. In *SPAA: Proceedings of the Seventh Annual ACM Symposium on Parallel Algorithms and Architectures*, ACM Press, Santa Barbara, California, pp. 95–105.

Barker, K. J., and Kerbyson, D. J. 2005. A performance model and scalability analysis of the HYCOM ocean simulation application. In *IASTED PDCS*, IASTED, Phoenix, Arizona.

Bell, C., Bonachea, D., Cote, Y., Duell, J., Hargrove, P., Husbands, P., Iancu, C., Welcome, M., and Yelick, K. 2003. An evaluation of current high performance networks. In *International Parallel and Distributed Processing Symposium*, IEEE Press, Nice, France, pp. 10.

Bell, C., Bonachea, D., Nishtala, R., and Yelick, K. 2006. Optimizing bandwidth limited problems using one-sided communication and overlap. In *International Parallel and Distributed Processing Symposium*, ACM/IEEE Press, Rhodes Island, Greece, pp. 10.

Chen, W., Iancu, C., and Yelick, K. 2005. Communication optimizations for fine-grained UPC applications. In *International Conference on Parallel Architectures and Compilation Techniques*, ACM/IEEE Press, Saint Louis, Missouri, pp. 267–278.

Coarfa, C., Dotsenko, Y., Eckhardt, J., and Mellor-Crummey, J. 2003. Co-Array Fortran performance and potential: An NPB experimental study. In *International Workshop on Languages and Compilers for Parallel Computing*, Springer Berlin / Heidelberg, College Station, Texas, vol. 2958 of *Lecture Notes in Computer Science*, pp. 177–193.

Danalis, A., Kim, K., Pollock, L., and Swany, M. 2005. Transformations to parallel codes for communication-computation overlap. In *Supercomputing Conference*, ACM/IEEE Press, Seattle, Washington, pp. 58.

Dunigan, T. H., Vetter, J. S., White, J. B., and Worley, P. H. 2005. Performance evaluation of the Cray X1 distributed shared-memory architecture. *IEEE Micro vol. 25*, no. 1, pp. 30–40.

Halliwell, G. R. 2004. Evaluation of vertical coordinate and vertical mixing algorithms in the hybrid coordinate ocean model (HYCOM). *Ocean Modelling vol. 7*, no. 3–4, pp. 285–322.

Hiranandani, S., Kennedy, K., and Tseng, C. 1992. Compiling Fortran D for MIMD distributed-memory machines. *Communications of the ACM vol. 35*, no. 8, pp. 66–80.

Hoisie, A., Lubeck, O., and Wasserman, H. 2000. Performance and scalability analysis of teraflop-scale parallel architectures using multidimensional wavefront applications. *International Journal of High Performance Applications vol. 14*, no. 4, pp. 330–346.

HPCS. High Productivity Computing Systems initiative in DARPA. See `http://www.darpa.mil/ipto/programs/hpcs/`.

Iancu, C., Husbands, P., and Hargrove, P. 2005. HUNTing the overlap. In *International Conference on Parallel Architectures and Compilation Techniques*, ACM/IEEE Press, Saint Louis, Missouri, pp. 279–290.

Jones, P. W., Worley, P. H., Yoshida, Y., J. B. White III, and Levesque, J. 2005. Practical performance portability in the parallel ocean program (POP). *Concurrency and Computation: Practice and Experience vol. 17*, no. 10, pp. 1317–1327.

Ke, J., Burtscher, M., and Speight, E. 2005. Reducing communication time through message prefetching. In *International Conference on Parallel and Distributed Processing Techniques and Applications*, CSREA Press, Las Vegas, Nevada, pp. 557–563.

Ke, J., Burtscher, M., and Speight, E. 2005. Tolerating message latency through the early release of blocked receives. In *Euro-Par*, Springer Berlin / Heidelberg, Lisbon, Portugal, vol. 3648 of *Lecture Notes in Computer Science*, pp. 19–29.

Kerbyson, D. J., and Jones, P. W. 2005. A performance model of the parallel ocean program. *International Journal of High Performance Computing Applications vol. 35*, no. 3, pp. 261–276.

Kerbyson, D. J., Alme, H. J., Hoisie, A., Petrini, F., Wasserman, H. J., and Gittings, M. 2001. Predictive performance and scalability modeling of a large-scale application. In *Supercomputing Conference*, ACM/IEEE Press, Denver, Colorado, pp. 39.

King, C. T., Chou, W. H., and Ni, L. M. 1990. Pipelined data parallel algorithms-I: Concept and modeling. *IEEE Transactions on Parallel and Distributed Systems vol. 1*, no. 4, pp. 486–499.

Koch, K. R., Baker, R. S., and Alcouffe, R. E. 1992. Solution of the first-order form of the 3-D discrete ordinates equation on a massively parallel processor. *Transactions of the American Nuclear Society vol. 65*, pp. 198–192.

Lawry, W., Wilson, C., Maccabe, A. B., and Brightwell, R. 2002. COMB: A portable benchmark suite for assessing MPI overlap. In *IEEE International Conference on Cluster Computing*, IEEE Press, Chicago, Illinois, pp. 472.

Leu, J., Agrawal, D. P., and Mauney, J. 1987. Modeling of parallel software for efficient computation-communication overlap. In *Fall Joint Computer Conference*, IEEE Press, Washington DC, pp. 569–575.

Liu, G., and Abdelrahman, T. S. 1998. Computation-communication overlap on network-of-workstation multiprocessors. In *International Conference on Parallel and Distributed Processing Techniques and Applications*, CSREA Press, Las Vegas, Nevada, pp. 1635–642.

MVAPICH/MVAPICH2. See `http://nowlab.cse.ohio-state.edu/projects/mpi-iba/`.

Quinn, M. J., and Hatcher, P. J. 1996. On the utility of communication-computation overlap in data-parallel programs. *Journal of Parallel and Distributed Computing vol. 33*, no. 2, pp. 197–204.

Sohn, A., Ku, J., Kodama, Y., Sato, M., Sakane, H., Yamana, H., Sakai, S., and Yamaguchi, Y. 1996. Identifying the capability of overlapping computation with communication. In *International Conference on Parallel Architectures and Compilation Techniques*, ACM/IEEE Press, Boston, Massachusetts, pp. 133.