On the Interference of Communication on Computation in Java

Barbara Kreaseck[3] Larry Carter[1] Henri Casanova [1,2] Jeanne Ferrante[1]
[1] Department of Computer Science & Engineering [2] San Diego Supercomputer Center
University of California at San Diego
[3] La Sierra University
{kreaseck, carter, casanova, ferrante}@cs.ucsd.edu

Abstract

Overlapping communication with computation is a well-known technique to increase application performance. While it is commonly assumed that communication and computation can be overlapped at no cost, in reality, they do contend for resources and thus interfere with each other. In this paper we present an empirical quantification of the interference rate *of communication on computation. We measure this rate on a single processor communicating with both local and remote processors via Java sockets. Among other results we find that the computation rate can suffer by as much as 50%, and that the reduction is approximately proportional to the communication rate. We conclude that interference deserves further study.*

1. Introduction

A common technique for improving the performance of a parallel application running on a distributed platform is to use simultaneous computation and communication (e.g., via multi-threading). However, overlapping computation with communication is not necessarily without cost, as the two can *interfere* with each other. Most published works (see for instance [1, 2, 3, 4, 5, 11, 12, 13]) have traditionally ignored the specific cost of this interference. By "interference" we do not mean the overhead of synchronization nor the idle time of a processor waiting for data; instead, we mean the reduction in computation speed caused by concurrent communication of independent data. This paper is a quantitative study of interference, which we hope will lead to better distributed algorithms.

Suppose processor A is concurrently computing and sending data to processor B. The transfer rate between A and B can vary, for instance because of network contention. We can measure the computation rate when different transfer rates are sustained. Figure 1 shows results for one of our testbed configurations (see Section 3 for details). The y-axis is the computation rate for processor A, normalized to the maximum achieved. The x-axis is the observed transfer rate in MB/sec. The graph shows that as the transfer rate increases the computation rate decreases in an approximately linear fashion. This behavior holds for all systems we have examined and we thus propose the following definition of the interference between computation and communication.

Definition: Given a set of measurements of the normalized computation rate at observed transfer rates, let l be the line which is the least squares fit of these observations. Then, the *Interference Rate of Communication on Computation* (IR) is the negative slope of l.

For Figure 1, the line l is $y \approx -0.037x + 0.96$, and thus IR $= 0.037$. An IR value of 0 implies that communication and computation may be perfectly overlapped, as assumed in most literature. However, in our experiments, there was always some level of interference.

This paper presents an empirical study of the interference of communication on computation for both near and distant heterogeneous processors. Within the framework of mostly Intel processors and Java with natural threads, we find that

- the computation rate can be reduced by over 50% when communication reaches maximum transfer rates, regardless of the number of communicating threads,
- the IR of receiving (IR_r) from a processor is generally larger than the IR of sending (IR_s) to the same processor, and
- when sending and receiving concurrently, IR_s and IR_r are different (usually smaller) than for one-way communication.

Our experiments are conducted in the limited scope of the Java Virtual Machine (JVM) in which the overlap of

This paper is based upon work supported by the National Science Foundation under Grant No. 0234233.

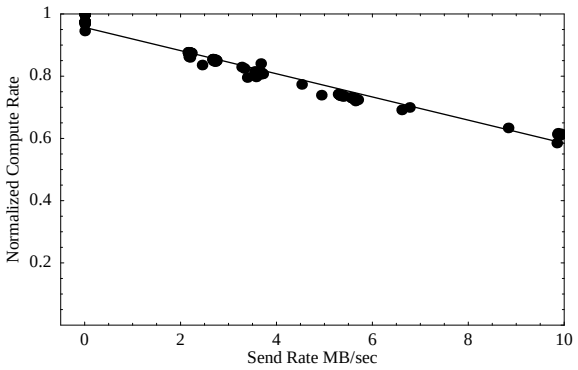


Figure 1. Interference Rate of Communication on Computation – Observed computation and transfer rates when Processor A computes and concurrently sends to processor B. The line shown is the least-squares fit on the plotted points. $IR_s = 0.037$, the negative slope of the line.

communication with computation is dictated by the JVM thread scheduling policy. Nevertheless this scope is significant as Java is used for large distributed peer-to-peer computing projects such as XtremWeb [6]. Our broader objective is to show that incorporating the interference rate of communication on computation could lead to better scheduling decisions for such systems. Indeed, in our previous work we studied the distribution of independent identical tasks to the nodes of a tree overlay network. We showed that, if the goal is to maximize throughput of the system, an optimal distribution can be computed [4]. Furthermore, our simulations verified that autonomous scheduling algorithms can achieve this optimum [11]. While our results hold for various processor models, all these models assume either full or no overlap of communication with computation. The empirical results presented in this paper suggest that none of the models exactly fits reality, but a new model can be used in practice (at least for Java-based systems).

In the next section we outline the method and scope of our work. We present our results in Section 3. Finally, in Sections 4 and 5 we discuss related work and conclude.

2. Methodology

2.1. Experimental framework

Each processor in our testbed can be configured to perform activities (compute, send, receive) in different threads concurrently. Most processors were running Java (j2sdk1.3.1 or later) with natural threads, and processors sent and received information through Java sockets.

We created a benchmark to stress both the compute rate and the transfer rate at a node. The atomic compute task repeatedly calculates k diagonals of the square of a matrix. Unless otherwise noted, the matrix was 1024×1024 integers (i.e., 4MB), and k=1. We measure the time it takes to compute each atomic compute task, and record its inverse, the compute rate in tasks/sec. The send and receive activities involved repeatedly sending or receiving 1MB of random data. The transfer rate is recorded in MB/sec.

Table 1 describes the nodes used in our study. Most are Intel-based desktop workstations running some flavor of Linux. A few are Suns running Solaris. The distance between nodes ranged from a few meters (in the same lab) to thousands of kilometers (across continents). As far as possible, we used quiescent systems and averaged results over at least 5 duplicate experiments.

2.2. Experimental scenario

All experiments in this paper measure the Interference Rate (IR) of communication on computation for the node Lab0. For a portion of each experiment (usually the beginning), Lab0 computes tasks in isolation. The rest of the experiment introduces activities involving other nodes, including communication with Lab0. By correlating the compute and transfer rates, we are able to determine the IR of communication (send and/or receive) on Lab0's computation.

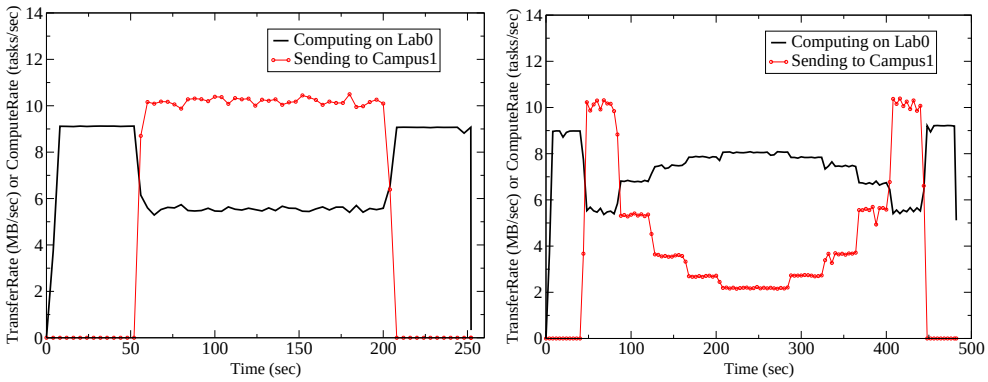
We designed our experiments to display a variety of compute rates and transfer rates on Lab0. Figure 2(a) shows a simple experiment: computing on Lab0 and sending to Campus1. The x-axis shows the time in seconds. The y-axis shows both the transfer rate in MB/sec and the compute rate in tasks/sec. As seen in the graph, Lab0 starts sending to Campus1 at time $T = 50$ seconds, and stops sending at $T = 200$. Over the whole experiment, this provides two clusters of points: one cluster has communication rate 0 MB/sec and compute rate values around 9 tasks/sec; the other has values around 10 MB/sec and 5.5 tasks/sec.

Two points are not sufficient to show that the IR is *linearly* related to the communication bandwidth. Consequently, we introduce a varying load into the system. To vary the sending transfer rate from Lab0 to the receiving node, we introduce contention on the receiving node. Figure 2(b) shows an initial computation on Lab0 joined by a sending transfer to Campus1 at time $T = 40$ seconds. Starting at $T = 80$, different nodes start sending to Campus1. This added contention at Campus1 reduces the transfer rate from Lab0 to Campus1. The figure shows that as the sending transfer rate at Lab0 drops, its compute rate improves. These variations provide us with multiple observations of the compute and transfer rates.

Next, we convert the raw observations into related pairs of rates. To eliminate anomalies caused by the granularity of our measurements, we average the compute rates and

Label	Arch	OS	MHz	RAM MB	Location
Lab0	P4	Linux RH K2.4	1700	512	HPC Lab (SD) oh
Lab1	P2	Linux RH K2.4	300	512	HPC Lab (SD) tandem
Lab2	P4	Linux RH K2.4	1700	512	HPC Lab (SD) ct
Lab3	P4	Linux RH K2.4	2000	1024	HPC Lab (SD) pa
Lab4	U.SPARC	SunOS 5.8	440	256	HPC Lab (SD) kalmar
Lab5	U.SPARC	SunOS 5.8	333	128	HPC Lab (SD) picard
Lab6	P2	Linux RH K2.4	451	384	HPC Lab (SD) boltzmann
Campus0-8	dual P3	FreeBSD 4.6.2	800x2	1024	AW Cluster (SD) broach, et al.
Campus9	U.SPARC	SunOS 5.8	333	128	APE Lab (SD) ursus
SB0-4	P3 Xeon	Linux Deb. K2.4	2200	512	UCSB Mayhem (SB) ash, et al.
SB5	P4	Linux Deb. K2.4	1800	512	UCSB Mayhem (SB) charcoal
Tenn	P3 Cu.mine	Linux Deb. K2.4	700	320	SAU (TN) cpp
Brazil	P3 Cu.mine	Linux RH K2.4	865	640	UFCG (BR) lula
France	quad Xeon	Linux RH K2.4	2400x4	1024	ENS-Lyon (FR) graal

Table 1. Processor architecture, operating system, and physical location of nodes in our testbed. We measured computation speed on Lab0 while it communicated within our Lab and across Campus, California (SB), the U.S. (TN), the equator (BR), and the Atlantic (FR).



(a) Steady Transfer Rate

(b) Varying Transfer Rate.

Figure 2. The steady transfer rate in (a) results in two clusters of data points(transfer rate, compute rate) $\approx$ (0,9) or (10,5.5). In (b), while Lab0 sends to Campus1, other processors send to Campus1 for various intervals, creating bandwidth contention and varying the send transfer rate on Lab0.

transfer rates over regular intervals. After normalizing the compute rate to the maximum achieved compute rate, we graph the related points. For instance, Figure 1 shown earlier graphs the rate data from the experiment of Figure 2(b). Finally, we perform a least-squares linear fit and compute IR as the negative slope of this fit. Thus, the IR is the percentage decrease in the computation rate per every MB/sec of transfer rate. We also performed and report on experiments in which Lab0 receives data from other nodes, and in which Lab0 both sends and receives data.

3. Results

3.1. Impact of receiving on computing

As we saw in Section 1, there is a steady degradation in the computation rate as the data transfer rate increases. A linear fit describes the relationship quite well. We conducted experiments with a local node in our lab, Lab0, computing and receiving from one other node in Table 1. A single experiment generates an average of 95 data points relating observed transfer rates to normalized computation rates. For each experiment, we made a least squares fit. Over all 117 experiments, a *linear* fit errs by at most 9.1%; the average max error is 2.2% and the standard deviation is around 0.75%. A *quadratic* least-squares fit errs by at most 7.4% (average max error is 1.8%) with a standard deviation around 0.52%. The quadratic fit can better accommodate outliers at the extremes of the data transfer rates. Yet the accuracy of the linear fit is almost as good as the quadratic fit, and the linear fit is simpler.

The left half of Table 2 shows the Interference Rates due to receiving (IR_r) when Lab0 computes and receives from one other node. The average IR_r for Lab0 is approximately 0.052. This means that on average when Lab0 is receiving at 10 MB/sec, it degrades to $1 - 10 \times 0.052 = 48\%$ of

Node	Lab0 Receives from Node				Lab0 Sends to Node			
	avg IR_r	avg const	avg StDev	avg MaxE	avg IR_s	avg const	avg StDev	avg MaxE
Lab1	**0.0458**	0.9714	0.0125	0.0312	**0.0314**	0.9909	0.0085	0.0226
Lab2	**0.0467**	0.9770	0.0146	0.0396	**0.0299**	0.9740	0.0119	0.0247
Lab3	**0.0468**	0.9781	0.0136	0.0371	**0.0299**	0.9752	0.0124	0.0301
Lab4	**0.0452**	0.9674	0.0150	0.0371	**0.0199**	0.9922	0.0048	0.0155
Lab5	**0.0467**	0.9723	0.0212	0.0434	**0.0211**	0.9704	0.0249	0.0631
Lab6	**0.0465**	0.9754	0.0140	0.0376	**0.0305**	0.9738	0.0112	0.0276
Campus0	**0.0474**	0.9777	0.0053	0.0192	**0.0388**	1.0296	0.0184	0.0567
Campus1	**0.0477**	0.9823	0.0052	0.0163	**0.0373**	1.0223	0.0209	0.0547
Campus2	**0.0473**	0.9772	0.0048	0.0147	**0.0383**	1.0245	0.0186	0.0549
Campus3	**0.0471**	0.9757	0.0056	0.0255	**0.0381**	1.0219	0.0151	0.0509
Campus4	**0.0474**	0.9790	0.0054	0.0214	**0.0367**	1.0122	0.0185	0.0540
Campus5	**0.0475**	0.9777	0.0046	0.0155	**0.0377**	1.0218	0.0205	0.0554
Campus6	**0.0474**	0.9779	0.0050	0.0163	**0.0372**	1.0136	0.0157	0.0507
Campus7	**0.0472**	0.9765	0.0052	0.0167	**0.0377**	1.0177	0.0141	0.0410
Campus8	**0.0477**	0.9797	0.0048	0.0145	**0.0366**	1.0073	0.0148	0.0456
Campus9	**0.0494**	0.9737	0.0233	0.0905	**0.0237**	0.9745	0.0175	0.0404
SB0	**0.0510**	0.9844	0.0053	0.0171	**0.0310**	0.9771	0.0026	0.0098
SB1	**0.0515**	0.9840	0.0054	0.0150	**0.0312**	0.9781	0.0026	0.0124
SB2	**0.0513**	0.9844	0.0053	0.0153	**0.0313**	0.9762	0.0022	0.0071
SB3	**0.0511**	0.9841	0.0044	0.0144	**0.0310**	0.9758	0.0027	0.0127
SB4	**0.0511**	0.9758	0.0058	0.0147	**0.0307**	0.9762	0.0034	0.0158
SB5	**0.0501**	0.9762	0.0053	0.0163	**0.0307**	0.9784	0.0032	0.0110
Tenn	**0.0743**	0.9979	0.0019	0.0090	**0.0638**	0.9911	0.0016	0.0074
France	**0.0848**	0.9989	0.0024	0.0166	**0.0372**	0.9966	0.0015	0.0079
Brazil	**0.0891**	0.9992	0.0013	0.0053	**0.0431**	0.9955	0.0008	0.0037

Table 2. IR_r when Lab0 receives from various nodes. IR_s when Lab0 sends to various nodes.

its maximum compute rate. Depending upon the processor type, operating system, network connection and distance to Lab0, the IR_r varies between 0.0452 and 0.0891. The dominant factor is distance. For the closest nodes (Lab$n$) the IR_r is roughly 0.0463, and the IR_r increases with distance to above 0.074. For the France node, the receive transfer rates were so low (with or without contention) that the data looked more like a cluster than a line; nevertheless, the computed IR_r was consistent with other locations.

We also experimented with Lab0 receiving data from several nodes simultaneously. We observed a maximum total receiving transfer rate of just over 11 MB/sec on Lab0. When receiving from a single quiescent near node, the observed data transfer rate achieves 85%-95% of that maximum. When receiving from multiple quiescent near nodes, the observed data transfer rate achieves that maximum. In general, once a node is receiving at its maximum transfer rate, the *impact on computation* is maximized, regardless of the number of receiving threads.

It turns out that it is possible to estimate the resulting compute rate at Lab0 using the sum of the individual IR_r's weighted by their respective transfer rates. We calculate the *expected* normalized compute rate to be $[1 - \sum_{i=1}^{k}(IR_r(i) \times TR(i))]$ when receiving from k nodes, where $IR_r(i)$ is the interference rate from node i, and $TR(i)$ is the transfer rate from node i.

When computing on Lab0 and concurrently receiving from four nodes with similar IR_r values within the Lab, the expected compute rate varies from the actual compute rate by 2.58% on average. For dissimilar IR_r values, (e.g., when computing on Lab0 and concurrently receiving from Lab1, Campus4, SB1, and Tenn), the expected compute rate varied from the actual compute rate by 1.21% on average. Over all our experiments with multiple receives on Lab0, using the aggregate interference rates to predict compute rates yields a *standard deviation* as low as 0.22%, and 0.82% on average. This demonstrates the utility of collecting the individual interference rates (IR_r's) between nodes. These rates can be combined arbitrarily to understand the aggregate impact of receiving multiple communications on computation.

3.2. Impact of sending on computing

When a node is computing and concurrently sending to other nodes, we also see a steady degradation in the compute rate per MB/sec of sending. We conducted experiments with Lab0 computing and concurrently sending to a receiving node chosen from those listed in Table 1. Over all 118 IR_s experiments we performed, a linear least-squares fit errs by at most 9.6% (average max error is 3.1%) with a standard deviation around 1.05%. A quadratic least-squares fit errs by at most 10.1% (average max error is 2.6%) with a standard deviation around 0.74%. Although both of these

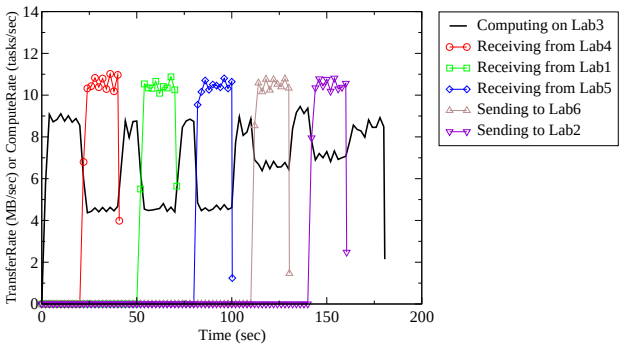


Figure 3. Impact of Receiving versus Sending on Computation.

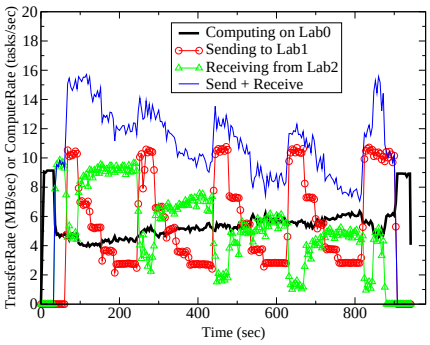


Figure 4. Example of a variety of bandwidth usage by time and its affect on computation.

fits are looser than those we saw with receiving, we still choose to use a linear fit for simplicity.

The right half of Table 2 shows the Interference Rates due to sending (IR_s) when Lab0 computes and sends to one other node. The average IR_s for Lab0 is around 0.034. This means that on average when Lab0 is sending at 10 MB/sec, its computation rate degrades to 66% of its maximum compute rate. Thus, *sending has less of an impact on computation than receiving does*. This is illustrated in Figure 3, which shows computing on Lab0 and a sequence of three receives followed by two sends, all involving different nodes. The three receives clearly show a larger interference on the compute rate than the two sends.

In our experiments, the IR_s varies between 0.0199 and 0.0638. With receiving, the distance between nodes had the greatest influence on the value of IR_r. However, with sending, the operating system also significantly influences the value of IR_s, as explained in Section 3.7.

In general, the greater the distance between nodes, the larger the interference rate. Our first results from sending to nodes within the Lab, across Campus, and across the state to Santa Barbara displayed this general principle. Our more current results (duplicative over two months and displayed within the table of this section) show an exception to that general rule. Notice that for sending, the Campus nodes actually have IR_s values that are larger than those of the SB nodes. The maximum achieved transfer rate from Lab0 to Campus is 10.2, while the maximum achieved transfer rate to SB nodes is only 6.4 (down from 7.5 in original runs). Also, the least-squares linear fit for the Campus nodes contain constant values above 1. We suspect a change in the underlying intervening network structures, but have no information in that regard.

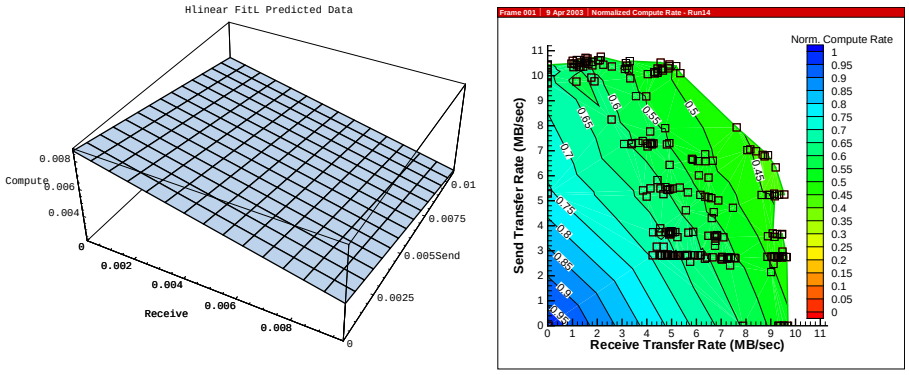
The observed maximum transfer rate for sending is slightly less than that for receiving. When simultaneously sending to six nodes, Lab0 achieved a rate of around 10.7MB/sec. As with receiving, once a node is sending at its maximum transfer rate, the impact on computation is maximized, regardless of the number of sending threads. Again, we can estimate the resulting compute rate using the formula $[1 - \sum_{i=1}^{k}(IR_s(i) \times TR(i))]$, where $IR_s(i)$ is the interference rate of sending to node i, and $TR(i)$ is the transfer rate to node i. When computing on Lab0 and concurrently sending to four Linux nodes with similar IR_s values within the Lab, the calculated compute rate varies from the actual compute rate by 0.88% on average. For dissimilar IR_s values due to distance, (e.g., when computing on Lab0 and concurrently sending to Lab1, Campus4, SB1, and Tenn), the calculated compute rate varied from the actual compute rate by 2.04% on average. Over all our experiments with multiple sends on Lab0, using the aggregate interference rates to predict compute rates yields a *standard deviation* as low as 0.43%, and 1.21% on average.

3.3. Combined impact of sending and receiving

One might expect that when a processor node is receiving at rate $TR(A)$ from node A and sending at $TR(B)$ to node B, the resulting normalized compute rate would be the weighted sum $1-[TR(A)\times IR_r(A)]-[TR(B)\times IR_s(B)]$. Interestingly, we found that this was not the case. We observed a synergistic effect, where the actual compute rates were higher than the formula above predicts.

Figure 4 shows one of the experiments we used to investigate the combined effect of sending and receiving on computation. Lab0 computes while sending to Lab1 and receiving from Lab2. Look first at the receiving rate from Lab2 marked by triangles. This rate starts at roughly 9 MB/sec and drops to 7, 5 and 4 as receiving contention on Lab2 is increased. Within each of these 200 second long periods, the sending rate, shown by the circles, has five levels. This gives us a rich collection of send and receive rates. The compute rate is shown by the heavy black line.

Having had success with a linear fit when sending or receiving alone, we performed a least-squares planar fit of the send and receive transfer rates to the compute rates. The resulting equation for the plane is $NormComputeRate \approx C - [TR(A) \times IR_r'] - [TR(B) \times IR_s']$, where IR_r' and



(a) Least Squares Planar Fit

(b) Contour graph of Compute rates.

Figure 5. Send rate, Receive rate, and Compute rate on the corresponding x-, y- and z-axes.

IR_s' are the coefficients for the interference rates of receiving and sending in combination, and C is the constant term, which is usually close to 1. For the data points from Figure 4, the fit equation is $NormComputeRate \approx 0.9542 - [TR(A) \times 0.0427] - [TR(B) \times 0.0265]$. Figure 5(a) shows the fit plane. The planar fit for this example differed from the measured data by at most 5.2%, with a mean error of 1.4% and standard deviation of 1.1%.

Interestingly, the interference rate of receiving alone from A is $IR_r = 0.0502$, and the interference rate of sending alone to B is $IR_s = 0.0327$. These are both higher than the interference rates when sending (IR_s') and receiving (IR_s') in combination. Thus sending and receiving in combination have a synergy; the compute rate observed when sending and receiving combined is higher than that predicted by sending or receiving alone.

Another way to view the impact of combined sending and receiving on computation is through a contour map. Figure 5(b) shows the raw data points and plots the compute rate as a contour map on the Send-Receive plane. The send rate is mapped along the y-axis and the receive rate is mapped along the x-axis. At the origin when both the receive rate and the send rate are zero, the compute rate is at its highest point. The contour lines are relatively straight, suggesting a linear relationship. If sending and receiving had equal interference upon computing, we would expect these lines to be at a -45% angle. Instead, we observe that they are closer to a -60% angle, showing that the interference from receiving is greater than that of sending, even when in combination.

Figure 5(b) also shows the synergy of combined sending and receiving on computation. Notice the data points where the sending rate is about 3 MB/sec. The corresponding receive rate varies between 4.5 and 9.5 MB/sec. Here, as the receive rate increases, the compute rate drops from 0.70 to almost 0.45, spanning almost 5 contour sections. Contrast this with the group of data points where the sending rate is about 10.5 MB/sec. We see that the corresponding receive rate varies between 1.0 and 5.5 MB/sec. The compute rate drops from 0.65 to 0.5, spanning 3 contour sections. From these two groups of data points we observe that the impact on computation is steeper at a sending rate of 3 MB/sec than at 10.5 MB/sec. Thus we can receive at a higher rate for a lower additional impact on computation when the send rate is high than when it is low. See [10] for extensive yet similar experimental results for other testbed configurations.

As in our example above, the interference rates from the planar fit on combined sending and receiving are lower than those predicted from sending and receiving alone. On average, the compute rate calculated from the planar fit coefficients had a maximum error of 19.1%, and a mean error of 1.5% with a standard deviation of 1.2%. If we ignore the synergy and use the IR_r and IR_s values from receiving or sending alone to predict the compute rate, we found on average, a maximum error of 37.3%, and a mean error of 5.3% with a standard deviation of 3.1%. As with our example, using IR_r and IR_s values from sending or receiving alone increases the mean error, but may still be within an acceptable range as a predictor.

3.4. Impact of computation data footprint

The use of cache and the TLB differs with the range of memory addresses used by a compute-intensive process. By changing the data footprint of the computational tasks in our experiments, we can see whether the cache and TLB usage impact the IR. To achieve different data footprints, we change the size of the matrix that we use as the basis of our computation task.

We varied the size of the matrix from 1024×1024 down to 8×8 for experiments in which Lab0 computes and receives data from Campus8. We found that as the matrix

size decreases so does the IR_r (from 0.049 down to 0.025). We hypothesize that as the task footprint fits into smaller levels of the memory hierarchy on Lab0, receiving has less of an opportunity to interfere with the computation's use of memory. We observed the same decrease in IR_s values. Since different applications have different data footprints, it is likely that for each application, the IR_r and IR_s values will need to be measured anew.

3.5. Impact of different message sizes

All experiments mentioned so far used a message size of 1 MB. We also investigated the effect of the size of the messages on the Interference Rates. For Lab0 computing and sending to Lab3, we experimented with message sizes from 8 MB down to 128 KB. The IR_r values differ by at most 0.0012. There is slightly more variation in the IR_s values, differing by at most 0.0020. Overall, we believe that the observed difference in interference rates show no significant variation with message size.

3.6. Near vs. remote nodes

The IR_r on the more remote nodes was in general higher than on near nodes. When the more remote nodes (Tenn, France, Brazil) were sending, they *achieved* IR_r values over 0.075 (see Table 2). Were they actually able to send at Lab0's maximum receive transfer rate, this would constitute an enormous obstacle to computation. In reality, they were only able to send at rates well below 1 MB/sec, and thus lowered the computation rate by at most 7.5%.

The near nodes had lower IR_r values, around 0.50, but were able to achieve a much higher send rate, approaching the maximum of 11 MB/sec. Thus, in practice, they lowered the computation rate by as much as 50%. When a SBn node was sending, its IR was similar to machines on Campus. However, it had a smaller actual impact due to its only being able to reach a send transfer rate of 7 MB/sec. In the long run, when receiving the same amount of data from near and more remote nodes, the more remote nodes will have a higher overall impact on computation: it will be a lower unit amount spread over a relatively longer time frame, so that the impact at any one time will be lower than that when receiving from a near node.

3.7. Differences by software at a node

The most striking difference by software occurs between nodes running Linux and those running Solaris. On a configuration where Lab0 is receiving from two near Linux and two near Solaris nodes, each node uses a relatively equal portion (roughly 2.8 MB/sec) of Lab0's receive bandwidth. Yet when Lab0 is sending to that same collection of nodes, the send data transfer rate is unequally divided. The Solaris nodes equally share a smaller portion of Lab0's send data transfer rate, around 1.8 MB/sec. The Linux nodes equally share a larger portion, around 3.8 MB/sec. While our goal is not to explain such behaviors, they had the following impact on our evaluation of the interference of communication on computation.

First, experiments involving the Solaris nodes (Lab4 and Lab5) resulted in significantly higher relative error of the linear fit than when only Linux nodes are used (roughly by a factor 4). Second, for sending, we observe that nodes running Solaris have lower IR_s values than other nodes at similar distances from Lab0. For nearby nodes (Lab$n$), those running some flavor of Linux have an IR_s around 0.030, while those running Solaris (Lab4 or Lab5) have IR_s values near 0.020. Similarly with the Campus$n$ nodes: the Solaris node, Campus6, has an IR_s of 0.0237 while the rest have values closer to 0.0370.

Not only does the operating system impact the interference of communication on computation but also the Java implementation at a node. In particular, some Java implementations are unable to use 'native threads' and instead must use 'green threads', which precludes any overlapping of communication and computation. In our experiments, we ensured that all our computing nodes ran Java with natural threads.

3.8. Two-way communication

Frequently, the communication between machines occurs in both directions. One reason would be to return the results of the task to the originating machine. Another could be that machines are producing/consuming dependent information. Regardless, we measured the interference rate on computation on one machine that was both sending and receiving with another machine.

For most near machines, the contributing interference rate of both sending and receiving was higher. Together, neither machine was able to operate at full bandwidth, yet each was able to achieve a higher bandwidth than had they been performing the same type of communication (either both sending or both receiving). This is another case of the synergy of concurrently sending and receiving.

4. Related Work

We present work on the impact of communication on computation. Most published work (see for instance [1, 2, 3, 4, 5, 11, 12, 13]) makes the simplifying assumption that communication and computation may overlap without cost. This work, which is part of a larger work [10] that studies dynamic autonomous scheduling on large-scale heteroge-

neous systems, provides empirical evidence that challenges the above assumption.

Several works deal with the overlap of communication and computation for particular hardware [7, 8, 15, 14]. By contrast our work evaluates the overlap capability between Java threads running in a JVM on mostly Intel processors running some flavor of Linux or Solaris.

We can note the work in [9], which decomposes communication into a fraction that can be overlapped by computation, and a fraction that cannot. Our empirical results could be used to instantiate such a model.

5. Conclusion

In this paper we present an empirical study of the interference of communication on computation for multithreaded Java. We define the *interference rate of communication on computation* (IR) to be the negative slope of the linear least-squares fit representing the relationship between the rate of computation on a processor and the rate(s) of communication between processors. We deployed an experimental measurement system on a select group of both near and distant heterogeneous processors. Within the framework of mostly Intel processors and Java with natural threads, we found that the computation rate is reduced by over 50% when communication reaches its maximum transfer rates. This reduction is roughly linear with the amount of data transferred per second, and is independent of the number of communicating threads. Furthermore, the IR of receiving (IR_r) from a processor is generally larger than the IR of sending (IR_s) to the same processor. Among other results, an intriguing one was that there is a synergy between sending and receiving, meaning that the IRs of concurrent communication, IR'_r and IR'_s, are smaller than IR_r and IR_s. Our results should be considered when developing algorithms that attempt to improve performance by overlapping computation and communication, at least in the context of distributed computing systems implemented with Java.

References

[1] T. Abdelrahman and G. Liu. Overlap of computation and communications on shared-memory networks-of-workstations. *Journal of Parallel and Distributed Computing Practices*, 2(2):145–153, June 1999.

[2] S. B. Baden and S. J. Fink. Communication overlap in multi-tier parallel algorithms. In *Proceedings of SC'98, Orlando Florida*, November 1998.

[3] F. Baude, D. Caromel, N. Furmento, and D. Sagnol. Optimizing metacomputing with communication-computation overlap. *Lecture Notes in Computer Science*, 2127:190–204, 2001.

[4] O. Beaumont, L. Carter, J. Ferrante, A. Legrand, and Y. Robert. Bandwidth-centric Allocation of Independent Task on Heterogeneous Platforms. In *Proceedings of the 16th International Parallel and Distributed Processing Symposium (IPDPS'02)*, 2002.

[5] V. Bouchitte, P. Boulet, A. Darte, and Y. Robert. Evaluating array expressions on massively parallel machines with communication/computation overlap. *International Journal of Supercomputer Applications and High Performance Computing*, 9(3):205–219, 1995.

[6] G. Fedak, C. Germain, V. N'eri, and F. Cappello. XtremWeb: A Generic Global Computing System. In *Proceedings of the IEEE International Symposium on Cluster Computing and the Grid (CCGRID'01)*, May 2001.

[7] G. Goumas, A. Sotiropoulos, and N. Koziris. Minimizing completion time for loop tiling with computation and communication overlapping. In *Proceedings of the 15th International Parallel and Distributed Processing Symposium (IPDPS-01)*, page 39, Los Alamitos, CA, Apr. 23–27 2001. IEEE Computer Society.

[8] S. K. S. Gupta, C.-H. Huang, P. Sadayappan, and R. W. Johnson. A technique for overlapping computation and communication for block recursive algorithms. *Concurrency: Practice and Experience*, 10(2):73–90, Feb. 1998.

[9] K. Högstedt, L. Carter, and J. Ferrante. On the parallel execution time of tiled loops. *IEEE Transactions on Parallel and Distributed Systems*, 14(3):307–321, March 2003.

[10] B. Kreaseck. *Dynamic Autonomous Scheduling on Heterogeneous Systems*. PhD thesis, University of California at San Diego, 2003.

[11] B. Kreaseck, L. Carter, H. Casanova, and J. Ferrante. Autonomous Protocols for Bandwidth-Centric Scheduling of Independent-task Applications. In *Proceedings of the 17th International Parallel and Distributed Processing Symposium (IPDPS'03)*, 2003.

[12] A. Lain and P. Banerjee. Techniques to overlap computation and communication in irregular iterative applications. In *Proceedings of the International Conference on Supercomputing, Mancester, England*, pages 236–245, 1994.

[13] M. J. Quinn and P. J. Hatcher. On the Utility of Communication-Computation Overlap in Data-Parallel Programs. *Journal of Parallel and Distributed Computing*, 33(2):197–204, 1996.

[14] A. Sohn and R. Biswas. Communication studies of dmp and smp machines. Technical Report NAS-97-005, NASA Ames Research Center, March 1997.

[15] A. Sohn, J. Ku, Y. Kodama, M. Sato, H. Sakane, H. Yamana, S. Sakai, and Y. Yamaguchi. Identifying the capability of overlapping computation with communication. In *Proceedings of the 1996 Conference on Parallel Architectures and Compilation Techniques (PACT '96)*, pages 133–138, Boston, Massachusetts, Oct. 20–23, 1996. IEEE Computer Society Press.