Message Strip Mining Heuristics for High Speed Networks

Costin Iancu[1], Parry Husbands[1], and Wei Chen[2]

[1] Computational Research Division, Lawrence Berkeley National Laboratory
{cciancu,pjrhusbands}@lbl.gov
[2] Computer Science Division, University of California at Berkeley
wychen@cs.berkeley.edu

Abstract. In this work we investigate how the compiler technique of message strip mining performs in practice on contemporary high performance networks. Message strip mining attempts to reduce the overall cost of communication in parallel programs by breaking up large message transfers into smaller ones that can be overlapped with computation. In practice, however, network resource constraints may negate the expected performance gains. By deriving a performance model and synthetic benchmarks we determine how network and application characteristics influence the applicability of this optimization. We use these findings to determine heuristics to follow when performing this optimization on parallel programs. We propose strip mining with variable block size as an alternative strategy that performs almost as well as a highly tuned fixed block strategy and has the advantage of being performance portable across systems and application input sets. We evaluate both techniques using synthetic benchmarks and a hand-optimized application kernel from the NAS Parallel Benchmark Suite.

1 Introduction

Reducing the overhead of message transfers is the goal of many optimization techniques for parallel programs. One such strategy, message vectorization [19], reduces communication time by hoisting fine-grained reads and writes outside loops and coalescing them into block transfers. This has the benefit of speeding up communication both by amortizing startup costs and by taking advantage of the higher bandwidths realized for larger messages. Vectorization has been implemented in optimizing compilers for a variety of parallel programming languages [9, 14, 13], and is widely recognized as an extremely useful manual optimization for message-passing programs.

While message vectorization is a common and effective optimization, it alone is not enough to minimize total run time, as the processor remains idle while the network is busy performing the bulk message transfer. In this paper we focus on a technique called *message strip mining* [18] that may further enhance the effectiveness of the message vectorization optimization. While a vectorized loop waits for the remote memory access to complete before it proceeds with local computation, message strip mining divides communication and computation into sub-blocks and pipelines their executions by skewing the loop. This leads to an increase in the number of messages sent and thus message startup costs, but has the potential of reducing communication overhead through the overlapping of non-blocking send and receive operations with independent computation. Techniques such as loop unrolling can further enhance the effectiveness for message strip mining by increasing the number of operations available for overlap.

Applying message strip mining carelessly may result in performance degradation due to both increased message startup costs and network contention. Furthermore, performance is directly influenced by application characteristics; the data transfer size and the ratio between communication and computation affect the amount of available overlap. A systematic scheme is therefore required to evaluate, for a vectorizable loop, whether it is worthwhile to perform message strip mining and loop unrolling based on the network parameters and application communication patterns. Our contributions in this paper are: 1) *An enumeration of application and machine characteristics that influence the applicability and performance of message strip mining;* and 2) *A determination of how these characteristics guide us in choosing a good message decomposition for message strip mining based on analytical models and experimental results.*

Our work differs from previous investigations [17, 18] in that we consider both network and application characteristics, use a performance model based on LogGP [6], and use both synthetic and application benchmarks that capture the characteristics of a wide class of programs. We also introduce a variable block size message decomposition that performs in practice almost as well as the fixed size strategy while being less sensitive to variations in the performance parameters. Experimental results suggest that message strip mining is a highly effective optimization for any vectorizable loop whose total data transfer size exceeds a minimal threshold that has a typical value of 2-4 KBytes for today's high performance networks. Furthermore, the optimization does not require a large amount of local computation to be effective, provided the fetched remote data is loaded at least once; the overhead of the associated cache misses is usually enough to hide most of the communication time. Therefore, we believe message strip mining holds great potential as an automatic compiler optimization for reducing application communication costs.

The rest of this paper is organized as follows. Section 2 introduces message strip mining and identifies network and application characteristics that affect the success of this optimization. The analytical model for the performance of strip mining is described in Section 3, and our experimental results are presented in Section 4. Related work is surveyed in Section 5 and we conclude in Section 6.

2 Message Strip Mining

Strip mining is a loop transformation commonly associated with vectorizing compilers. In this context a single loop is first transformed into a doubly nested loop and the inner loop is replaced by a sequence of vector instructions. This transformation is illustrated in Figures 1 and 2 (the generation of vector instructions is omitted). The parameter S, often dependent on the size of a vector register is usually referred to as the *strip size*.

`for(i=0; i<N; i++)`
`  a[i] = b[i]+r[i];`
	`for(i=0; i<N; i+=S)`
	`  for(ii=i; ii<min(i+S-1,N), ii++)`
	`    a[ii] = b[ii]+r[ii];`
		`get(lr, r, N);`
		`for(i=0; i<N; i++)`
		`  a[i] = b[i]+lr[i];`

Fig. 1. Unoptimized loop.	**Fig. 2.** Strip mined loop.	**Fig. 3.** Vectorized loop.

In this paper we explore a less well-known application of strip mining: its ability to reduce the communication overhead of parallel programs. Following the standard "owner computes" rule that most parallel paradigms adhere to, we consider only remote read operations as candidates for message strip mining, though the analytical model and optimization techniques presented in this paper can easily be adapted to support remote writes.

Returning to Figure 1 and assuming that `a` and `b` are local arrays and `r` is remote we now demonstrate message strip mining. Figure 3 displays the results of applying message vectorization to the unoptimized loop. Performance is significantly improved by copying all remote value in one bulk transfer instead of performing a read operation in every iteration. One disadvantage with such a transformation, however, is that the processor must wait for the completion of the remote transfer (denoted by $T(N)$) before proceeding with the local computation. Figure 4 demonstrates the process of message strip mining, which attempts to reduce this communication cost. The single bulk transfer[3] from Figure 3 is divided into several blocks based on the strip size, and the loop is then skewed so that both the communication and computation code can be performed in a pipelined manner. This transformation is thus similar in spirit to software pipelining, as it also exploits the parallelism within the loop body by allowing the communication and computation phases of several iterations of the loop to be processed simultaneously. Moreover, the computation is not the only source of overlap; as Figure 5 shows, loop unrolling can be additionally applied to allow the communication operations for different strips to be issued at the same time and increase the amount of overlap (both computation and communication) available in the unrolled loop body. In the ideal scenario (ignoring the overheads of initiating and completing remote operations), the communication time of each strip is completely overlapped with independent computation or communication from previous iterations, leaving only the overhead $T(S)$ of transferring the strip peeled off the loop body. Since in general the strip size is much smaller than to the total data size, the optimal performance gain $T(N) - T(S)$ can be significant. A less obvious benefit of this optimization is that it can help reduce the amount of *shadow* memory used to store remote elements. While a vectorized loop requires an N element buffer, the message strip mined loop can reuse the

[3] We denote a blocking memory read operation by `get(dest, src, nbytes)` and a non-blocking read operation by `h = nbget(src, dest, nbytes)`. The completion test for a non-blocking remote memory operation is denoted by `sync(h)`.

shadow memory and therefore only requires $U*S$ elements, where U is the unroll depth of the loop.

```
h0 = nbget(lr, r, S);
for(i=0; i<N; i+=S)
   h1 = nbget(lr+S*(i+1), r+S*(i+1), S);
   sync(h0);
   for(ii=i; ii<min(...); ii++)
      a[ii] = b[ii]+lr[ii];
   h0=h1;
```

Fig. 4. Message strip mining.

```
h[0] = get(lr, r, S);
for(i=0; i<N; i+=S*U)
    h[1] = nbget(..., S);
    ...
    h[U] = nbget(..., S);
    sync(h[0])
    for(ii=i; ii<S; ii++)
        a[ii] = b[ii]+lr[ii];
    ...
    sync(h[U-1]);
    for(ii=i+S*(U-1); ii<min(...); ii++)
        a[ii] = b[ii]+lr[ii];
    h[0] = h[U];
```

Fig. 5. Unrolling a strip mined loop.

2.1 Practical Considerations for Message Strip Mining

Message strip mining decomposes the transfer of data into a series of smaller transfers overlapped with local computation, thereby lowering the overall runtime of the application. Loop unrolling exposes more potential for overlap by increasing the number of messages that can be simultaneously issued. Both optimizations, however, have the potential for causing performance degradation: message strip mining by increasing the startup cost of communication and loop unrolling by increasing message contention in the network. To understand the impact of combining these transformations, one has to take into account both machine characteristics and application characteristics.

Machine Characteristics: The LogGP [6, 10] network performance model approximates the cost of a data transfer as the sum of the costs incurred by the transfer in all system components. This is illustrated in Figure 6. Parameters of special relevance to us are o_s and o_r, the send and receive overhead of a message; G, the inverse network bandwidth; and g, the minimal gap required between the transmission of two consecutive message.

According to the model, the cost of a single message transfer can be divided into two components; the software overhead on both the send and receive side, as well as the time the message actually spends in the network. The total communication cost of the vectorized loop presented in Figure 3 is thus $T_{\text{vect}}(N) = o + G * N$, where o is the sum of o_s and o_r. The total communication cost of the strip mined loop (in Figure 4), on the other hand, must be expressed as the sum of the time spent issuing the non-blocking *gets* and the time spent waiting for the corresponding *syncs*. It is now easy to see how message strip mining could have a negative impact on performance. Although the transformation does not increase message volume, it incurs more software overhead due to more messages being issued. Furthermore, since smaller messages are transferred this overhead becomes a greater fraction of communication time. While picking a small strip size allows for more overlapping of the waiting times, it is also accompanied by an increase in issue time. Loop unrolling introduces further complications, as the effective time to issue a non-blocking *get* operation can depend not only on the transfer size S but also on the number of these operations issued in sequence U. Hence large unroll depths, though theoretically attractive, may not be desirable in practice due to this network limitation. In Section 3 we analyze these tradeoffs in greater detail.

Another potential source of performance degradation is introduced by network resource limitations. The LogGP model assumes an "ideal" network with no resource constraints and considers the parameters to be constant. In practice, combining loop unrolling with message strip mining increases the number of outstanding messages on the Network Interface Card at a given time and can have a large performance impact on networks where resources are scarce.

In addition to NIC resources, remote DMA message transfers also compete with the processor for access to a node's memory system. This "interference" may adversely affect the speed of local computation by effectively increasing the time to access memory. These effects are considered further in Section 4.2.

Application Characteristics: Data transfer size is perhaps the most important factor in determining the effectiveness of message strip mining for a vectorized loop. An intuitive rule of thumb is that each strip transfer should be a bandwidth-bound ($G * N > o$) message, so that the increase in message startup costs can be compensated for by the potential performance gain from hiding the message transfer time. We further explore this concept in Section 4.3.

Even with a large transfer size, message strip mining is still not useful unless we can discover enough computation to overlap. In other words, the computation cost of a strip, $C(S)$, should not be significantly smaller than its communication cost $T(S)$. While loop unrolling mitigates this problem by increasing the number of iterations that execute simultaneously, it also leads to an increase in message initiation time. Since network performance is generally orders of magnitude worse than CPU performance , it may appear that sufficient overlap could not be attained without a large amount of computation. This assumption is not true, however, as it neglects the cost of memory access. Because remote data is typically transferred (by DMA) directly to main memory (and not the cache), they must then be brought into the cache for further processing. Cache miss penalties, which are also incurred by the vectorized loop, provide more time for the overlap of communication with computation. Thus, when estimating the "effective" local computation cost, one must also take into account the effect of local memory bandwidth. We study the implications further in Section 4.1.

Finally, the application's communication pattern determines the degree contention in the network system and so influences the effectiveness of our transformation. We consider the following communication patterns, ranging from least to most contention: *one-to-one* and *all-to-one*.

The rest of the paper is devoted to finding heuristics to determine for any given system the following parameters: the minimum amount of computation that benefits from message strip- mining, the minimum message size where message strip-mining begins to pay off and the optimal message decomposition and schedule.

3 Estimating the Impact of Strip-Mining

We begin our analysis by using the LogGP model we derive the communication costs for the loops shown in Figures 1, 3, 4, and 5.

Individual Small Messages (Figure 1): $T_{\text{small}}(N) = N * \text{EEL}$. One is forced to pay the full network cost for a small message on every access.

With Message Vectorization (Figure 3): $T_{\text{vect}}(N) = o + G * N$. This is the cost of one large message transfer.

Message Strip-Mining and Unrolling (Figures 4 and 5):

In this case each *sync* waits if the transfer (with time $G * transfersize$) is not completed by the time the execution encounters the *sync*. The total communication cost is the sum of the overhead of initiating all the transfers and the waiting time for the $s$ync call of each transfer. If the message is divided into m blocks $S_1, ...S_m$, the waiting time $W(S_i)$ for each block can be expressed as an equation based on the network parameters and computation costs:

$$\begin{aligned}
W(S_1) &= G * S_1 - \text{issue}(\text{S}_2) \\
W(S_2) &= G * S_2 - C(S_1) - W(S_1) - \text{issue}(\text{S}_3) \\
\ldots \quad &\quad \ldots \\
W(S_m) &= G * S_m - C(S_{m-1}) - W(S_{m-1}) \qquad (1)
\end{aligned}$$

Each equation is in addition subject to the constraint $W(S_i) >= 0$. The above represents the case where the unroll depth is one; equations for different number of unrolls can be derived in a similar fashion. $C(S_i)$ refers to the costs of computation performed on a block, and for simplicity we assume it can be expressed as a linear function $K * S_i$, without much loss of generality[4]. As the goal of message strip mining is to minimize the communication cost, it is equivalent to solving the above system to find m and the sequence $S_1, \ldots, S_m$ that minimizes the objective function

$$T_{strip+unroll} = \sum_{i=1}^{m} \text{issue}(\text{S}_\text{i}) + \text{W}(\text{S}_\text{i}) \qquad (2)$$

Finding the optimal message decomposition involves exploiting the tradeoffs on m, the number of transfers. A small m may minimize the waiting time but cause an excessive startup overhead, while a large m may exhibit the opposite problem and a similar tradeoff is encountered when varying the unroll depth. In the next section we analytically develop message decomposition heuristics that are straightforward to implement yet can achieve significant performance gains over a vectorized loop.

[4] We are interested in balanced algorithms where communication and computation are of the same order of magnitude.

3.1 Message Decomposition

Our analysis focus on loops with $K \approx G$ as the performance of strip mining for computation bound algorithms is less sensitive to the message decomposition due to increased potential for overlap. A simple yet effective decomposition is to make each block equal-sized, so that the total communication cost for a given loop only depends on one variable[5]. To find the optimal block size S under this scheme, we simply search sample decompositions with a representative amount of computation. Our synthetic experiments in the next section detail our findings.

While the fixed block size scheme is straightforward, it may not achieve optimal message overlap. In particular, the fixed-size decomposition may cause the waiting times for the blocks to oscillate between two extremes. It can be best explained by examining Equation (1): the second sync can be completely overlapped with the waiting and computation time of the first block, making $W(S_2)$ zero. This, however, means the next sync does not benefit from $W(S_2)$ at all (it can still be overlapped with its computation), resulting in a large $W(S_3)$, which in turn makes $W(S_4)$ free. It thus appears that for message strip mining, a sequence of varying block sizes that better captures the nature of the waiting time as recurrence relations may outperform the fixed-size scheme. As a heuristic, we pick the block sizes to be a geometric series $S_i = (1 + f)S_{i-1}$, where f is between 0 and 1. As a starting block size we choose the size of the smallest message that still benefits from strip mining as described in Section 4.3. A good value of f is determined mostly by application characteristics, and is clearly critical to the performance of our varying block size heuristic; we discuss in Section 4.5 heuristics for determining a lower bound for the values of the parameter f.

4 Experiments

In this section we provide quantitative data for the issues discussed in previous sections on the systems detailed in Table 1. The benchmarks are implemented in Unified Parallel C [5] and run over a customized communication layer [4, 8] with performance very close to that of each system's native communication API. The first three subsections discuss the effects that memory and network subsystems have on our optimizations. In particular, we determine the amount of computation and the minimal total transfer size required for message strip mining to be effective. Our next set of experiments attempts to determine the best message decomposition for the different combinations of network and application communication and computation pattern. For the fixed-size scheme we identify the optimal block size, while for the variable block size scheme we search for the parameter f. Finally, we illustrate the effectiveness of message strip mining by performing the optimization on the NAS Multigrid application kernel.

[5] We assume U is also fixed; certainly optimal block size can vary with the unroll depth

System	Network	CPU type
IBM Netfinity cluster [2]	Myrinet 2000	866 MHz Pentium III
IBM RS/6000 SP [1]	SP Switch 2	375 MHz Power 3+
Compaq Alphaserver ES45 [15]	Quadrics	1 GHz Alpha

Table 1. Systems Used for Benchmarks

4.1 Estimating the Cost of Computation

Our model in the previous section suggests that the effectiveness of message strip mining heavily depends on whether the vectorized loop contains a sufficient amount of computation that can be used to hide the communication latency. As mentioned in Section 2, the architecture's effective local memory bandwidth plays a major role on the overall computation cost required by the transferred data. In most of today's high performance computing systems the network is not tightly integrated with the memory hierarchy; the three platforms examined in this paper have their NICs attached either to a PCI bus (Myrinet and Quadrics) or to a proprietary bus (the SP switch), and two of them (Myrinet and Quadrics) use remote DMA operations that bypass the processor's cache. Accordingly, the cost of the computation that operates on the transferred data is composed of two parts: 1) cache miss penalties incurred by accessing the transfered data, and 2) execution time required by the computation itself. While the second component obviously varies from application to application, the cache miss penalty is an inherent part of the computation overhead and does not depend on the type of computation performed. As the performance gap increases between the processor and memory subsystem, the sustained memory bandwidth will likely become the bottleneck in determining the amount of computation available that can be overlapped with communication using message strip mining.

System	Inverse Network Bandwidth (μsec/Kb)	Inverse Memory Bandwidth (μsec/Kb)	Ratio (memory/network)
Myrinet	6.089	4.06	67%
Quadrics	4.117	0.46	11%
SPSwitch	3.35	1.85	55%

Table 2. Comparison on the network and memory data transfer rate.

Based on this observation, we compare the cost of moving data across the network with the overhead of moving the same data through the memory hierarchy. The inverse network bandwidth, or the G parameter of LogGP, is measured by timing the end-to-end latency of a large *get* transfer so that the communication cost is dominated by the wire latency. To determine the latency of moving data across the memory system, we measure the execution time of a code sequence that performs an integer vector reduction `accum += data[i];`. In some sense, this represents the minimal amount of computation possible on the transferred data (one memory access, one integer add, and one address increment), assuming the each remote element is referenced at least once. and it also has the lowest memory overhead due to the unit stride access.

Table 2 presents the results for our test platforms. Memory access time proves to be an important source of computation overhead. For some of our test platforms, the cost of fetching data from main memory is at least half the cost of moving the same data from network, with the exception of the Quadrics system. This provides an important insight, that essentially no limitation exists on the minimal amount of computation a vectorized loop must exhibit before it could benefit from message strip mining; as long as the loop accesses all of its remote values, in general the cache miss penalties alone will provide adequate amount of computation that allows communication latencies to be effectively hidden.

4.2 Interference Effects

To determine the effect of network transfers on local computation, we time simple computation loops on a node where the networking subsystem is serving remote read requests from a variable number of processors. As local computation we use the loop presented in the previous section that accesses memory with stride one and a loop performing the same vector addition but using indirect accesses with a uniform index distribution. The latter exhibits a much higher processor to memory traffic.

For all systems, our results show a slowdown of the computation loop ranging from 3% to 6%. The slowdown increases with the increase in local memory traffic and it is not affected by the number of nodes reading data from the memory space of the computation node. We conclude that interference from the network DMAs does not substantially affect the cost of memory access, and therefore does not substantially affect the effectiveness of strip mining.

4.3 Networking Subsystem Influence

In this section we use the LogGP model to guide us in determining the minimum message size that benefits from message strip mining. Based on our model, we note that any strip mining strategy must pay a minimum cost of $o+\max(o,g)+\epsilon$, because it decomposes the transfer into at least two messages. We include an extra ϵ to account for any syncs with a non-zero waiting time. For message strip mining to be worthwhile, we must thus have $o + G * N > o + \max(o, g) + \epsilon$, which implies $N > \frac{\max(o,g)+\epsilon}{G}$; a lower bound for N is $\frac{\max(o,g)}{G}$. Using values for the ratio $\frac{g}{G}$ obtained in [7] this lower bound is between 1KB and 3KB for all platforms. This value is verified by experiments described in the next section, where message strip mining performs well only for transfers larger than 2KB.

4.4 Influence of Application Characteristics

In order to determine the validity of our model and the “real-world” performance of message strip mining, we used a set of synthetic and application benchmarks. We chose the grid re-distribution step in the NAS-MG [3] benchmark to illustrate the point that memory traffic (in this case data redistribution) can provide

Network	No. Threads	Base	Strip Mining
Myrinet	2	1.24 (1)	0.81 (1.53)
	4	0.71 (1)	0.49 (1.45)
SP Switch	2	0.69 (1)	0.42 (1.64)
	4	0.44 (1)	0.35 (1.25)
Quadrics	2	0.32 (1)	0.28 (1.14)
	4	0.29 (1)	0.28 (1.03)

Table 3. Communication time for MG with different optimizations. The timing units are in seconds, and the parenthesized number reflects the speedup compared to the base case.

enough computation for overlap. We also use a synthetic benchmark with a problem configuration where each processor holds a variable number of double precision floating point values and performs either a reduction operation or a *vector-to-vector* operation. The total number of elements per processor is varied between 2^8 and 2^{20} with resulting total transfer sizes between $2KB$ and $8MB$. For the *vector-to-vector* operation, each element update involves three floating point operations and both the source and destination vectors are accessed with stride one.

For each benchmark, we provide implementations corresponding to the strip-mining strategies outlined in Figures 4 and 5. We vary the parameters N - total problem size, S - strip size, U - unroll depth, P - number of processors and the communication pattern. In evaluating the various message strip-mining techniques we seek to determine how much each technique buys us compared to the basic vectorization strategy. As such, in this section we present performance results showing the ratio of time taken by the "optimized" strategy to the vectorized strategy.

For lack of space, we have selected only the results presented in Figure 8. The results for the NAS-MG benchmark are presented in Table 3. The overall conclusion of this study is that while message decomposition with a fixed size strategy slightly outperforms the variable size strategy, its performance is more sensitive to the total transfer size, message size, communication pattern and the amount of computation on the transferred data. In practice, tuning the performance of the variable size strategy based only on the computation is enough to guarantee performance close to the best observed case for a fixed size. Unrolling loops more than two to four times is not a worthwhile optimization and there exists a lower bound on the message size that should be taken into account. The following sections explore into further detail the major trends observed across the machines

Effect of Block Size and Unroll Depth: Referring again to Figure 8 we discuss some of the major trends observed in the experiments. As a first observation, we note that a tuned strip mining implementation always improves performance compared to a vectorized-only implementation. The more expensive the computation following a network transfer, the better the impact of strip mining due to a higher opportunity for overlap. For example, on the IBM-SP for the *one-to-one* communication pattern we observe a 30% improvement in the best case for the “Reduction” benchmark compared to a 40% improvement in the best case for the “Vector” benchmark. Similar trends are observed for the other platforms.

For the *one-to-one* communication pattern we note the following: 1) Comparing the performance of the variable strip size strategy with the fixed size strategy we find that the latter outperforms the former (in the best case) for most problem sizes. For example, for the tuned Myrinet implementation of the variable size strategy we found the difference in performance in the range 0%-4%. Running the same implementation on a different platform, increases the relative difference to 10%-15%. For all platforms and communication/computation combinations, a fixed size message decomposition usually improves performance. For all platforms, there exists an optimal range for the message decomposition that depends on the total transfer size. Given the optimal decomposition, continuously decreasing the transfer size will result in slow-down compared to the vectorization case. The “optimal” message size range where a fixed decomposition outperforms a variable size decomposition is narrow. Furthermore, values outside this optimal range cause a performance degradation larger than the maximum of 15% observed for a badly tuned variable size implementation. 2) Combining strip mining with unrolling usually improves performance. On Myrinet and the IBM SP network, unrolling with small factors (2 or 4) improves performance. On the Quadrics network, unrolling causes a performance degradation.

For the *all-to-one* communication pattern we note: 1) While strip mining is still an effective strategy, increasing the degree of contention causes a decrease of speedup. 2) The fixed size strategy outperforms the variable size strategy (for a tuned implementation) by 2%-4%. Increasing the degree of contention causes an increase of the size of the optimal decomposition compared to the *one-to-one* communication pattern. 3) Loop unrolling does not improve performance for this communication pattern.

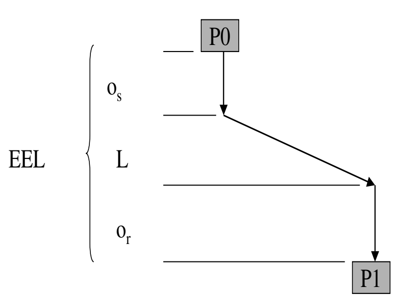

Fig. 6. Traditional LogP model for sending a message from processor P0 to processor P1.

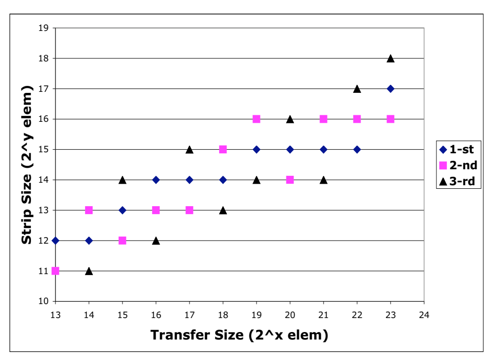

Fig. 7. Variation of the optimal fixed strip size with the total transfer size for Myrinet.

4.5 "Optimal" Message Decomposition

Based on our experimental results, we now present some guidelines for choosing an"optimal" message decomposition for a given problem. Our results indicate that the performance of a fixed size decomposition depends on the *total problem size*, on the *computation* and on the application *communication pattern*. Accordingly, when using this strategy, an application needs to be tuned with respect to all three parameters.

One important parameter for the fixed strip size decomposition is the ratio $\frac{N}{S}$, the total number of blocks transferred. Figure 7 presents the best three decomposition values based on the total message size for the "Reduction"benchmark on the Myrinet platform. Choosing the "optimal" decomposition for any given application can be achieved employing a search based strategy using the findings in this paper to prune the parameter search space. As a starting point for the search we choose the middle of the decomposition domain. We note that increasing the total size increases the block size. Increasing the degree of contention also increases the block size. Increasing the amount of computation decreases the strip size. However, as indicated by Amdahl's Law, the benefits of strip mining will be less and less pronounced with an increase in the cost of computation.

While employing a fixed size strategy, any variation of the three parameters requires a "re-tuning" of the application. On the other hand, the performance of the variable strip size strategy requires only an approximation of the computation cost. The performance of this approach is highly dependent on the values of the multiplication coefficient $1 + f$. This value determines the number of messages issued and the size of each message. For small values of f, the algorithm issues an increased number of transfer requests and thus incurs a high message initiation and control overhead. Increasing the value of f increases the probability of poor overlap since transfers might have to wait for completion. For best performance, a search based strategy is also required to determine the value of this parameter. In this paper we obtained a value of 0.45 by tuning the Myrinet implementation. This value performed as well on the IBM-SP platform and a little worse on the Quadrics platform. We have also experimented with "heavy" computations and found that the same value performs well enough in practice due to the diminishing returns implied by Amdahl's Law. We recommend for the Myrinet and the IBM-SP platforms a search range of 1.4 : 1.6 with an increment of 0.01. For the Quadrics platform we recommend a range of 1.1 : 1.4. In both cases, finding a good value for f requires a low number of experiments.

5 Related Work

Wakatani and Wolfe [18, 17] introduce message strip mining and analyze its impact for array redistribution in HPF and a code that implements a simple inspector-executor. They consider only strip mining with constant block size and use a simple model tailored for the array redistribution problem to predict the performance benefits.

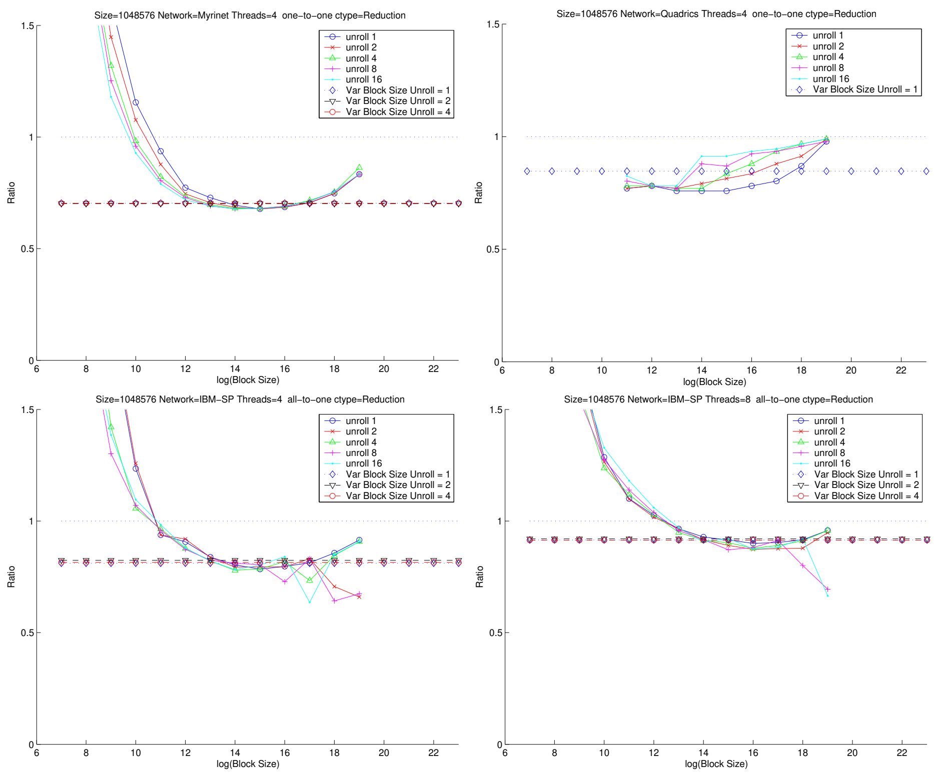

Fig. 8. Selected performance results. Problem size = 2^{20} doubles, ctype = Reduction.

Heras et al. [11] analyze vectorization techniques for Gaussian elimination codes on a Fujitsu VP2400/10 vector computer. They consider a combination of loop unrolling, interchange, fusion, distribution and sectioning techniques (strip mining, tiling and blocking). They do not analytically model the interactions among their optimizations and conclude that loop vectorization optimizations can be decoupled from other serial loop transformations since they do not modify the data reuse in the algorithm.

Gupta and Banerjee [12] present a methodology for estimating the communication costs in HPF programs. They use the analysis to guide the data partitioning decisions of their compiler and also to select communication primitives. Their work is directly applicable to the analysis of UPC programs, but it does not take into account contention as generated by the program communication pattern or the interference caused by the communication primitives on the memory performance of the communication peers. Evidence presented by Prieto et.al.[16] suggests that memory interference on the communication peers can severely affect performance.

6 Conclusion

In this paper we investigated message strip mining as a communication optimization technique, introduced strip mining with a variable size, demonstrated the effectiveness of both techniques and analyzed the factors that influence the performance of these program transformations.

We find the potential performance gain to be heavily dependent on both the network characteristics and the application characteristics. We empirically determine a lower bound on the total problem size after which our optimizations are effective. This value is 2KB for all networks studied. While we find that a well tuned fixed size strategy usually outperforms the variable size strategy by 0%-4%, its performance is very sensitive to all optimization parameters. On the other hand, the performance of the variable size strategy is determined only by the computation pattern and we recommend this approach when developing performance portable applications.

Besides application development, the heuristics that we examine in this paper are also of interest when developing collective communication libraries and vectorizing compilers. We plan on using the variable strip size strategy in our UPC implementation work in both areas.

Acknowledgment

The authors would like to thank Paul Hargrove for the GasNet program that determines network parameters.

References

1. IBM SP – Seaborg. http://hpcf.nersc.gov/computers/SP/.

2. NERSC Alvarez Cluster. http://www.nersc.gov/alvarez.
3. The NAS Parallel Benchmarks. Available at http://www.nas.nasa.gov/Software/NPB.
4. The UPC Runtime Specification, v 1.0 . Available at http:/upc.lbl.gov/docs/system.
5. UPC Language Specification, Version 1.0. Available at http://upc.gwu.edu.
6. A. Alexandrov, M. F. Ionescu, K. E. Schauser, and C. Scheiman. LogGP: Incorporating Long Messages into the LogP Model for Parallel Computation. *Journal of Parallel and Distributed Computing*, 44(1):71–79, 1997.
7. C. Bell, D. Bonachea, Y. Cote, J. Duell, P. Husbands, P. Hargrove, C. Iancu, M. Welcome, and K. Yelick. An Evaluation of Current High-Performance Networks. In *Proceedings of 17th International Parallel and Distributed Processing Symposium (IPDPS)*, 2003.
8. D. Bonachea. Gasnet specification, v1.1. Technical Report CSD-02-1207, University of California at Berkeley, October 2002.
9. Z. Bozkus, A. Choudhary, G. Fox, T. Haupt, and S. Ranka. A compilation approach for Fortran 90D/HPF compilers on distributed memory MIMD computers. In *Proceedings of the Sixth Workshop on Languages and Compilers for Parallel Computing*, Portland, OR, 1993.
10. D. E. Culler, R. M. Karp, D. A. Patterson, A. Sahay, K. E. Schauser, E. Santos, R. Subramonian, and T. von Eicken. LogP: Towards a Realistic Model of Parallel Computation. In *Principles Practice of Parallel Programming*, pages 1–12, 1993.
11. D. B. Heras and M.J. Martin and M. Amor and F. Arguello and F.F. Rivera and O. Plata. Comparing Vectorization Techniques for Triangular Matrix Decomposition Computations. In *5th International Conference on Parallel Computing*, September 1995.
12. M. Gupta and P. Banerjee. Compile-Time Estimation of Communication Costs on Multicomputers. In *Proceedings of the 6th International Parallel Processing Symposium*, Beverly Hills, CA, 1992.
13. M. Gupta, S. Midkiff, E. Schonberg, V. Seshadri, D. Shields, K.-Y. Wang, W.-M. Ching, and T. Ngo. An hpf compiler for the ibm sp2. In *Proceedings of the 1995 ACM/IEEE conference on Supercomputing (CDROM)*, page 71. ACM Press, 1995.
14. S. Hiranandani, K. Kennedy, and C.-W. Tseng. Compiling Fortran D for MIMD distributed-memory machines. *Communications of the ACM*, 35(8):66–80, 1992.
15. Lemieux. http://www.psc.edu/machines/tcs/lemieux.html.
16. M. Prieto and I. M. Llorente and F.Tirado. Data Locality Exploitation in the Descomposition of regular Domain Problems. In *IEEE Trans. on Parallel and Distributed Systens*, volume Vol 11, pages 1141–1150, 2000.
17. A. Wakatani and M. Wolfe. A New Approach to Array Redistribution: Strip Mining Redistribution. In *Proceedings of PARLE'94 (Athen, Greece)*, Jul 1994.
18. A. Wakatani and M. Wolfe. Effectiveness of Message Strip-Mining for Regular and Irregular Communication. In *PDCS (Las Vegas)*, Oct 1994.
19. H. Zima and B. Chapman. Compiling for distributed-memory systems. In *Proceedings of the IEEE*, 1993.