Hiding Communication Latency in Data Parallel Applications

Vivek Garg and David E. Schimmel
School of Electrical and Computer Engineering
Georgia Institute of Technology
Atlanta, GA 30332-0250
e-mail:{vivek,schimmel}@ece.gatech.edu

Abstract

Interprocessor communication times can be a significant fraction of the overall execution time required for data parallel applications. Large communication to computation ratios of the tasks performed by these applications results in suboptimal performance when executed on data parallel architectures. We present an alternate architectural framework, referred to as concurrently communicating SIMD (CCSIMD), which maintains the SIMD execution model, while introducing a small degree of task parallelism to exploit the communication concurrency. We introduce three different implementations of our architectural framework, and illustrate their effect on a suite of data parallel applications. Results show that CCSIMD architectures can provide a cost-effective way to hide communication latency in data parallel applications that can result in an increase in the performance of these applications.

1.0 Introduction

Communication in parallel computing represents the process of reading or writing data residing on a remote entity. In fine-grained SIMD machines, applications typically consist of interleaved phases of computation and communication. Consequently the execution time and hence performance of the applications is heavily dependent on the ability to efficiently carry out the communication as well as the computational power of the processing elements. During communication, the computational units in SIMD machines are forced to be idle, leading to a loss of efficiency. As a result, communication latency is often considered overhead. Minimizing this latency can lead to higher utilization of computing resources, which in turn can increase efficiency and performance.

The goal of this work is to explore alternatives for efficient communication in the SIMD class of machines, and investigate and implement mechanisms which exploit available concurrency to reduce communication latency for both regular and irregular communication. We propose a framework to exploit this concurrency, which we call concurrently communicating SIMD (CCSIMD) and will show that it can be very effective for a wide range of data parallel applications. This architectural framework can enhance the execution time of data parallel applications, and hide the communication latency incurred during their execution. Furthermore, these benefits can be obtained with a relatively small investment in hardware resources in both the control unit and the PEs.

1.1 Related Work

Several previous data parallel architectures have implemented *limited* support for concurrent computation and communication. Examples architectures include the Goodyear MPP [9], SLIM [4], and EXECUBE [8]; however, our proposed CCSIMD architecture provides a more general framework.

2.0 Parallel Communication in SIMD

A typical SIMD communication network is controlled centrally by the control unit, transmits messages synchronously, and has circuit-switched links. The communication channels utilized in the network are half-duplex. SIMD algorithms are executed in a lock-step manner, and as a result all active PEs execute the instruction being broadcast by the controller. The lock-step execution limits the PEs to either execute a communication or a computation operation, but not both. Since no overlap between computation and communication is possible, the PE architecture is designed to share hardware resources between computation and communication operations. An example of this is the MasPar MP-1 PE floating point unit which uses the mantissa unit for serially shifting the messages in or out of the PE during communication [3].

2.1 Message properties

A message set is a collection of communication primitives originating from all the PEs in the active set[1] and terminating at other active or passive PEs in the processor array. We define a message set to be *simple* if the routing of its member messages does not produce any contention for physical links. A *complex* message set, on the other hand, results in contention for physical links. A physical link set corresponding to a message set is defined to be the set of physical links required to route that message set. We define two message sets to be *mutually exclusive* if the intersection of their physical links is null. Two message sets are *interfering* if they are not mutually exclusive. Two message sets are *conflicting* or *data-dependent* if one or more src-dest PE pairs are common to both message sets and the communicated data for those pairs is destined for the same variable. For a more formal treatment of these issues, see [5].

3.0 Communication Concurrency in Data

1. The membership of the active set represents PEs which are to execute the current instruction being broadcast by the control unit.

Parallel Applications

There are three types of communication concurrency which can be exploited: communication and computation overlap, inter-communication concurrency, and intra-communication concurrency. This parallelism is available with respect to both local and global parallel communication.

3.1 Concurrent Communication and Computation

Lock-step execution in SIMD requires that any execution and communication be performed mutually exclusively. This disallows the possibility of overlapping the computation and communication phases in the application. Introduction of autonomy in the communication network can enable simultaneous execution of both communication and computation operations barring any data dependencies. This would also require decoupling the communication and computation resources in the PEs.

Techniques such as loop unrolling and software pipelining may be necessary to enhance the communication and computation overlap in applications. Most SIMD algorithms are written to fit the current SIMD paradigm which requires strict lock-step execution. This implies that most programs have an implicit dependency between the communication and computation phases and vice versa. With the ability to hide the latency of communication, these programs can be manipulated by compiler and/or scheduling techniques to transform the data dependencies and hence decouple the communication and computation phases of the algorithm.

3.2 Inter-Communication Concurrency

SIMD applications may have communication patterns that are specified in multiple phases. This implies that the SIMD instruction stream can have one or more consecutive communication operations. If the associated message sets are mutually exclusive, they may be routed at the same time. With the overlapping of multiple communications, say j, the effective time spent on communication for one phase can be reduced to

$$T_{\text{Comm}} = (max\{T^{i}_{\text{Comm}}\} + O_{\text{contention}} + j - 1)\forall(1 \le i \le j) \quad (1)$$

where $O_{\text{contention}}$ is the overhead incurred due to contention from multiple messages in the network. Depending on the significance of the incurred overhead, it may be possible to completely hide the communication times for all but the communication pattern with the longest routing time, and the time required to issue the other communication instructions in the network.

3.3 Intra-Communication Concurrency

Communication phases involving local communication are typically represented as a series of explicit communication steps specified with directional control by the programmer. As a result intra-communication concurrency does not exist in these patterns. Global communication on the other hand is typically specified as simply source destination pairs, and the global transport mechanism schedules the communication based on some routing protocol. We expect to find increased levels of intra-communication concurrency in global message patterns, since the message set corresponding to these communications can often be represented as a series of simple message sets with communication in all of the NEWS directions. This concurrency can be exploited to reduce latency for global communication significantly.

Consider the transpose communication pattern as a case study for this type of concurrency. A global router with full connectivity such as a crossbar may be able to route this pattern in a single cycle. Other types of global routers may take longer depending on the degree of connectivity and the router architecture. Yet another method of dealing with the transpose communication pattern is to decompose it into a series of non-conflicting message sets, as illustrated in Figure 1. The transpose in the figure has been decomposed into three phases, where in the ith phase the PEs i away from the diagonal exchange messages.

4.0 Concurrently Communicating SIMD (CCSIMD)

Concurrently Communicating SIMD (CCSIMD) enables concurrent execution of multiple communication instructions and overlapped execution of communication and computation. A CCSIMD architecture exploits this available concurrency by introducing communication autonomy in the PEs, and by separating the control of the communication and computation tasks. The main objective of CCSIMD machine is to hide the communication latency as much as possible by boosting communication instructions to overlap their execution with other computation and communication instructions.

The CCSIMD paradigm does not limit the number of message sets concurrently present in the network. The number of concurrent message sets in the network is bounded only by the communication concurrency in the SIMD code, and the implementation of the network. For example, consider two consecutive communication instructions in the code with corresponding message sets C_i and C_{i+1}. In SIMD execution, if C_i is injected into the network at time t, then C_{i+1} can not be injected into the network until time $t + \Delta t$, where Δt is the time required to transmit message set C_i. In CCSIMD the scheduling of these message sets depends on their inherent properties and their interaction with other message sets, i.e. whether the message sets are simple, complex, mutually exclusive, interfering, or conflicting. Mutually exclusive message sets can be injected into the network concurrently. Concurrent existence of interfering message sets in the network is only possible if the network can handle contention. Conflicting message sets coexisting in the network must complete in order, to ensure data consistency.

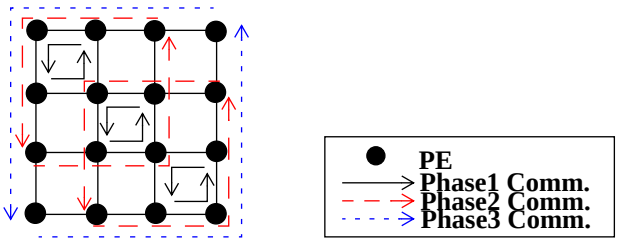

Figure 1. Decomposition of transpose into non-conflicting message sets.

4.1 CCSIMD Model

In this study we assume that there are two communication mechanisms: a 2-D toroid interconnection network and a global network. Communication is allowed in the four directions namely: North, East, West, and South (NEWS). Communication in the NEWS directions is allowed to proceed simultaneously, provided that there are no structural hazards. The interconnection network is used for communication resulting in simple message sets only, hence the communication times for the message patterns are deterministic. The number of links and channels per PE are a function of the degree of desired communication concurrency in the CCSIMD implementation. The global network is the only entity that can handle the irregular message patterns. However decomposition of complex messages sets into simple ones may be possible, and is encouraged for fast communication times. The global network communication times may be non-deterministic due to contention in the network. This is especially true for dynamically specified message patterns. The amount of contention and variability in the communication times is a function of the global network architecture.

Communication Instruction Support

There are two types of communication instructions available for local and global communication. All communication is register to register with timing very similar to that of the MasPar MP1/2 [3].

4.2 Issues in CCSIMD Design

The ability of CCSIMD to take advantage of communication concurrency leads to an increase in complexity of handling communication instructions, and additional hardware costs. Data dependencies must be analyzed to ensure conflict-free concurrent dispatch of communication and computation instructions. Furthermore, the combining of communication instructions in the network require scheduling and synchronization primitives, which may add to the cost and/or timing of the system.

Superscalar Dispatch

Concurrent execution of communication and computation instructions requires the controller to dispatch multiple instructions simultaneously. The control unit must now track the execution status and the state of the communication network concurrently. This requires addition of some hardware resources in the control unit to enable tracking of multiple message sets in the network, and to determine their completion. The PEs must also separate the communication and computation circuitry, resulting in the addition of buffer resources.

The dependency analysis can either be performed at compile time (static) or during execution (dynamic). Static analysis can be used to insert synchronization instructions in the code to ensure hazard-free handling of the instructions by the hardware. While this approach requires some extra time during compilation, it is relatively inexpensive in terms of hardware resources and its effect on the cycle time. Dynamic analysis requires dependency checking logic, but may be able to schedule more efficiently than the static case.

Multiple Message Sets

The decision to combine message sets in the network depends on the available bandwidth in the network, number of injection channels, and the register file bandwidth. The combination of communication instructions may be performed either in hardware or software or both. Combined communication instructions also require that the membership of the active PE set be updated to include the PEs corresponding to the instructions, and that the PEs know which communication to participate in.

Message sets in the network may be deterministic or non-deterministic. In the case of several deterministic message sets active in the network, the control unit can determine the completion of the message sets. However, some message sets may be destined to PEs further away than others. As a result, the control unit must track the status of each message set to determine when it has reached the destination. This is necessary to ensure timely ejection of message sets, and relinquishment of the network resources to incorporate other message sets.

Synchronization techniques for complex message sets is highly dependent on the routing and transmission mechanisms invoked by the network for such communication. The non- deterministic message sets require global synchronization schemes, since the control unit can not determine the completion time of the communication. Global OR has been traditionally used to perform such synchronizations, for example in the MasPar MP-2. For circuit oriented routing schemes, tear-down may be performed by the message tail flits, as in the CM-5 [10].

The ability to concurrently transmit multiple message sets in the PEs requires extra buffers and communication channels. The buffer requirements in a function of the number of dimensions and the maximum number of messages being transmitted via a PE. The flexibility of the PE to communicate with multiple directions also affects the switching complexity used for the communication buffers. The increase in communication channel resources can result in more wires and/or pins per PE.

4.3 Proposed Architectures

The CCSIMD framework can be implemented to exploit overlapped communication and computation concurrency, inter-communication concurrency, or even both. Our objective is to study several implementations exploring the possible variations of the amount of support built in for concurrent communication. In particular, we wish to study three different configurations.

- Type I: CCSIMD architecture built to exploit concurrency arising from the overlapped execution of communication and computation. This architecture requires a split control unit one each dedicated to handle communication and general execution. The PEs must be able to handle up to two instructions concurrently.
- Type II: CCSIMD architecture built to exploit inter-communication concurrency. While the control unit for this architecture does not have to be superscalar, it is required to efficiently dispatch consecutive communication instructions. As a result, the control unit is more complex than the SIMD control unit, but contin-

ues to operate as a single entity. The functionality of the PEs is the same as in a SIMD machine.

- Type III: CCSIMD architecture built to exploit the concurrency exploited in both Type I and II. Consequently the functionality of the control unit and the PEs is a combination of Type I and II architectures.

Both Type II and III architectures can also be implemented with varying degrees of support for concurrent communication. In particular, the number of concurrent communications in the network can range from one to four for a network with NEWS connectivity. This selection of configurations will help us understand the effect of exploiting these types of concurrency on data parallel applications. Implementation details of these architectures can be found in [5].

4.4 Comparison of Hardware Resources

Detailed analysis shows that the area of the PEs, the dominant component in a SIMD machine, needs to be increased by 1.6% and 5.1% of the MasPar SIMD PE area for Type I and Type II CCSIMD, respectively [5]. Thus, the additional hardware resources required for the CCSIMD implementations are quite modest, and as the machine size increase, the percentage increase in the system area will simply be reduced to the percentage area increase in the PEs.

5.0 Results

5.1 Comparison of Execution Times

The simulation data for the execution of the application suite on SIMD and the three types of CCSIMD architectures is presented graphically in Figure 2 along with the communication to computation ratios of the applications. As indicated, the fifth bar in each group represents the communication to computation ratio. A close look as this graph indicates that CCSIMD is very effective with respect to the overall execution times for those applications which have high communication to computation ratios. Cannon and reduce algorithms exhibit the highest amount of improvement over SIMD with 46% and 40%, respectively.

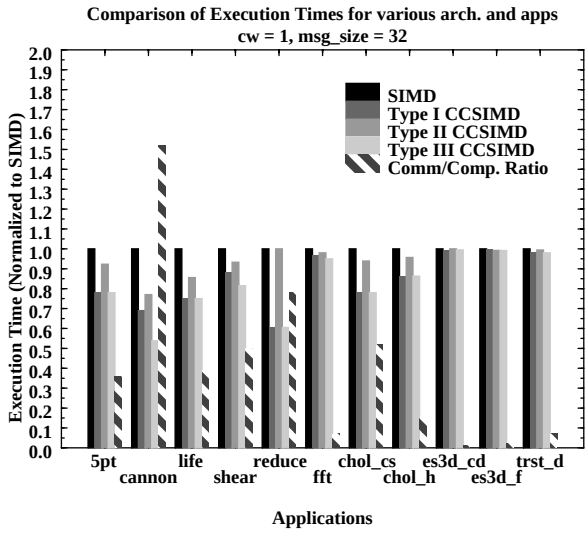

Figure 2. Comparison of simulated execution times for various applications on SIMD and CCSIMD architectures. Times are normalized to the SIMD execution time.

However, the primary goal of the CCSIMD architectures is to hide communication latency. The communication cycles listed for CCSIMD execution do not include those cycles where both communication and computation were executing concurrently. We consider these cycles to be hidden by computation, hence they are not considered a part of the communication cycles. The reduction in communication cycles ranges from a low of 9% to as high as 100%. Total elimination of communication overhead implies that all the communication was completely overlapped with computation, and as a result the communication latency is completely hidden. Figure 3 shows the effective communication cycles required during the execution of these applications. The bars represent the actual communication cycles normalized to the SIMD case. Note that all the applications including the ones which did not perform well in terms of execution time, are able to hide a significant amount of communication overhead under the CCSIMD execution. Also note that typically Type II architectures are less effective at reducing the communication overhead, however there may be applications where the data dependencies do not allow any overlapped communication and computation. In such applications, the best results are obtained from Type II architectures. There are several applications in the suite which have an added benefit from the use of Type II techniques in addition to the Type I overlap technique, of which Cannon, shear and fft clearly stand out. This implies that Type III architecture may be a candidate of choice for certain applications.

5.2 Communication Resource Utilization

In this experiment we determine the usage of the communication resources in the PEs as a fraction of the total communication cycles used. Figure 4 illustrates the number of cycles spent communicating in one, two, three, or all four directions. The data has been normalized to the number of communication cycles spent communicating in a single direction only. An inspection of this graph shows significant

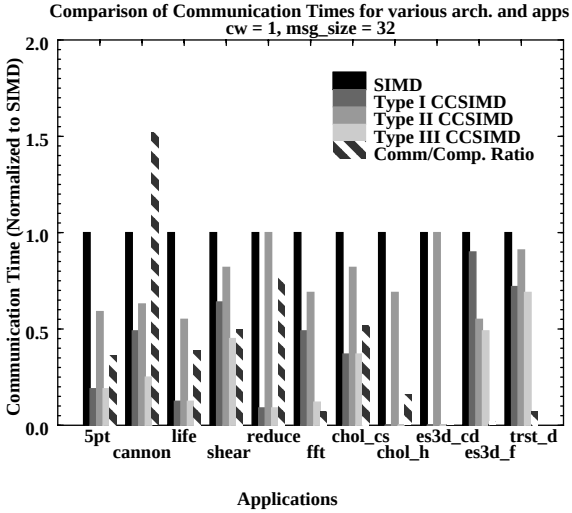

Figure 3. Comparison of simulated communication times for various applications on SIMD and CCSIMD architectures. Times are normalized to the SIMD execution time.

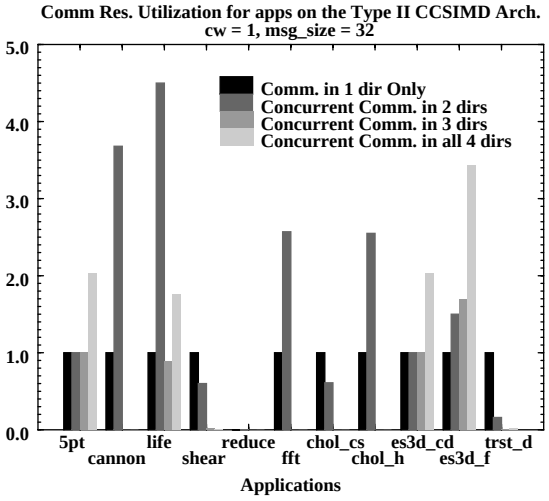

Figure 4. Concurrency in communication on the Type II CCSIMD architecture. The data shown on this graph has been normalized to the frequency of communication in a single direction only.

utilization of concurrent communication in all four directions. While four of the applications in the suite use the communication resources quite extensively, the remainder of the applications require concurrent communication in at most two directions. The performance improvement data presented in Figure 2 reveals that several of the applications which exhibit large gains from CCSIMD only require concurrent communication in two directions.

5.3 Effect of Channel Width on CCSIMD

We vary the channel width and observe its effect on the execution time of both SIMD and CCSIMD systems. We expect to see the benefits of CCSIMD diminish as the amount of communication time required on SIMD implementation goes down, i.e. as communication channel widths increase, the performance gains from CCSIMD diminish. Figure 5 shows the effect of varying communication channel widths on the execution times of the applications executing on the CCSIMD architecture. As expected, the best results for the CCSIMD architecture are seen at the lower communication channel widths. This is due to the relatively longer communication times at these widths. As the message size increases, we anticipate that the CCSIMD results for the larger channel widths will improve. CCSIMD architectures may require up to two times the communication channel and link resources. As a result a fair comparison of CCSIMD execution times with SIMD would require comparing the times for configurations which differ in their channel widths by a factor of at least two. In Figure 6 we show the largest channel width which yields lower performance in SIMD than the three types of CCSIMD architectures with a 1-bit communication channel. The channel widths differ among the different implementations of CCSIMD, and of course, from application to application.

Type I and Type III CCSIMD architecture outperform SIMD architectures with equivalent resources and for some applications even SIMD architectures with channel widths as high as 32 bits. This is primarily due to the overlapping of communication latency with computation time.

5.4 Effect of More Powerful PEs on CCSIMD

To examine the effects of employing powerful SIMD PE on the benefits of the Type I and II CCSIMD architectures, the trace for the application suite was regenerated based on these assumptions. The execution times for these applications, normalized to the SIMD time, are shown in Figure 7. A glance at this bar chart shows that indeed the Type II architecture now provides better execution times than the Type I architecture. Furthermore, the improvement in performance obtained by Type II architectures surpasses the best performance improvements for the some of the applications. Based on comparisons with the results with the original PEs shown in Figure 2, we can make several other observations. First we note that the shift in the balance of communication and computation certainly affects the com-

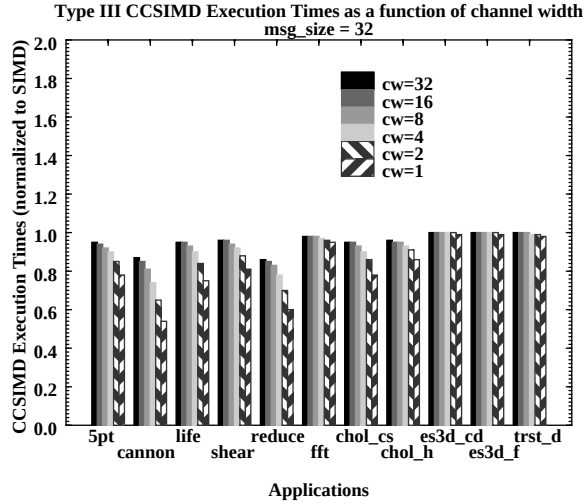

Figure 5. Type III CCSIMD execution times as a function of the communication channel widths.

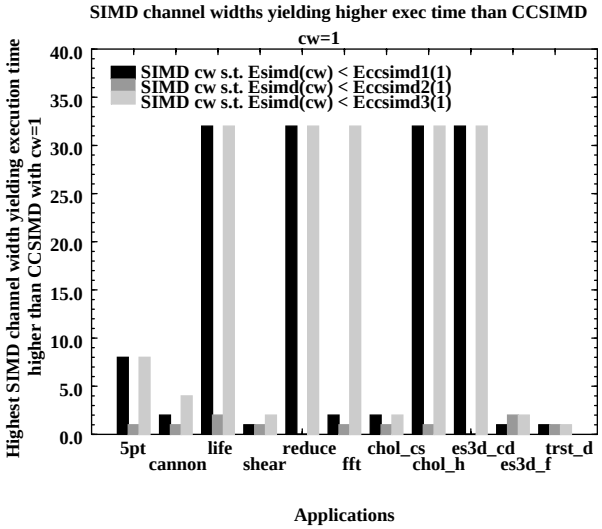

Figure 6. Comparison of the CCSIMD execution times with SIMD execution times. The bars represent the highest channel width in a SIMD configuration which under-performs a CCSIMD architecture with a channel width of 1-bit.

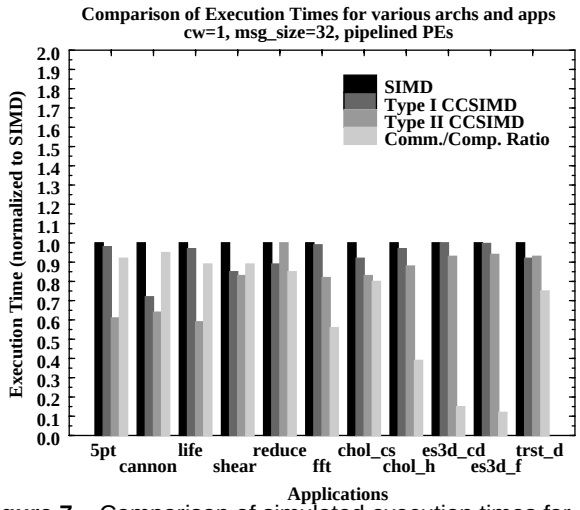

Figure 7. Comparison of simulated execution times for various applications on SIMD and CCSIMD architectures. Times are normalized to the SIMD execution time.

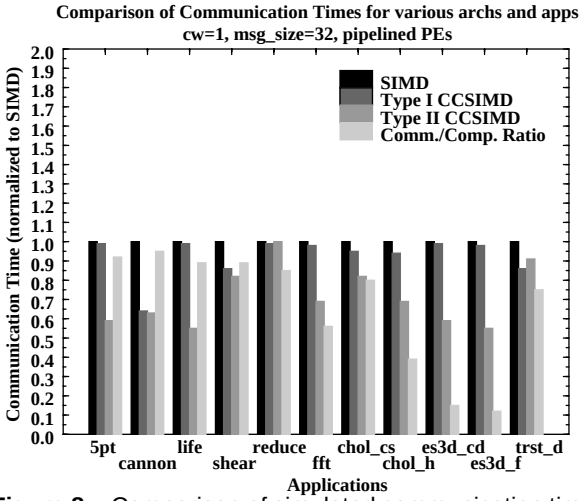

Figure 8. Comparison of simulated communication times for various applications on SIMD and CCSIMD architectures. Times are normalized to the SIMD execution time.

munication to computation ratio of these applications with the most dramatic change seen in the trst_d application whose communication to computation ratio increased from less than 10% to more than 70%. We also note that certain applications which experienced a rather insignificant improvement in performance previously, experience relatively better improvements in performance. This is due to the fact that communication latency with more powerful PEs becomes a more significant fraction of the overall execution time, and hiding communication latency provides more benefit than before.

Figure 8 shows the normalized communication times for the application suite. It is apparent from the graph that the communication latency hidden by Type II architecture is much higher than Type I unlike the previous results shown in Figure 3. This is due to the fact that the reduction in computation times results in a smaller number which can be overlapped with communication. Consequently, under these new assumptions, it is more advantageous to overlap communication with other communication. Of course, this is not always the case as illustrated by the trst_d execution times in Figure 7. Note that the execution time for the Type II implementation is slightly larger than that of Type I for the trst_d application. This fact is also reflected by the higher communication times for the Type II implementation for trst_d application in Figure 8. This application is unique in the sense that it exhibits a large communication to computation ratio, and yet the benefits obtained from CCSIMD are not significant. This is primarily due to the fact that the benefits obtained from CCSIMD are not just a function of the communication to computation ratio, but also depend heavily on the data dependencies within the applications.

6.0 Conclusions

Our experimental result show that CCSIMD architectures are certainly worth the modest investment in hardware resources required to implement them. The results indicate that for SIMD machines with relatively weak PEs, Type I and Type III architectures prove to be most beneficial. Furthermore, for our application suite the degree of communication concurrency necessary can be limited to two directions at a time. To generalize these results however, further experiments on other applications may be necessary. For machines with more powerful PEs, Type II architectures may prove to be more beneficial.

The benefits of CCSIMD architectures continue to increase as the problem size scales, which is apparent from Cholesky factorization study. For most applications, the CCSIMD architectures exhibit higher performance than SIMD machines with twice the channel widths or higher. This implies that the benefits of CCSIMD architecture can not be achieved by increasing resources on SIMD machines; instead a paradigm shift is necessary to gain the benefits available with CCSIMD.

7.0 References

[1] Allen, J.D. and Schimmel, D.E., "Issues in the Design of High Performance SIMD Architectures," *IEEE Transactions on Parallel and Distributed Systems*, 7(8), pp. 818–829, August 1996.

[2] Bjorstad, P.E. and Schreiber, R., "Unstructured Grids on SIMD Torus Machines," RIACS Technical Report 94.05, March 1994.

[3] Blank, T., "MasPar MP-1 architecture," *Proceedings of COMPCON Spring '90 - The Thirty-Fifth IEEE Computer Society International Conference*, San Francisco, CA, pp. 20-24.

[4] Chang, H.M., Sunwoo, M.H., and Cho, T.-H., "Implementation of a SLiM Array Processor," Proceedings of the 10th International Parallel Processing Symposium, Honolulu, Hawaii, April 15-19, 1996, pp.771-775.

[5] Garg, V., "Mechanisms for Hiding Communication Latency in Data Parallel Architectures," *Doctoral Dissertation*, http://www.ece.gatech.edu/users/vivek, Sept. 1997.

[6] Garg, V. and Schimmel, D.E., "CCSIMD: a Concurrent Communication and Computation Framework for SIMD Machines," *Proceedings of the Parallel Computer Routing & Communication Workshop* (PCRCW'97), Atlanta, Georgia, June 26-27, 1997, pp. 51-60.

[7] Kim, W. and Tuck, R., "MasPar MP-2 PE Chip: A Totally Cool Hot Chip," Fifth Hot Chips Symposium, August 8-10, 1993.

[8] Kogge, P.M. and Agrawal, P.A., "EXECUBE - a new architecture for scalable MPPS," P*roceedings of the 23rd International Conference on Parallel Processing*, Raleigh, N.C., August 15-19, 1994, vol. 1, pp. 77-84.

[9] Potter, J.L., Editor, *The Massively Parallel Processor*, The MIT Press, 1985.

[10] *The Connection Machine CM-5 Technical Summary*, Thinking Machines Corporation, October 1991.