Incomplete Information Processing for Optimization of Distributed Applications

Alfredo Cristóbal-Salas
School of Chemistry Sciences and Engineering, Autonomous University of Baja California, Tijuana, BC, Mexico, 22390,
cristobal@uabc.mx

Andrei Tchernykh
Computer Science Department, CICESE Research Center, Ensenada, BC, Mexico, 22830
chernykh@cicese.mx

Jean-Luc Gaudiot
Electrical Engineering and Computer Science, University of California, Irvine, USA, 92697
gaudiot@uci.edu

Abstract

This paper focuses on non-strict processing, optimization, and partial evaluation of MPI programs which use incremental data structures (ISs). We describe the design and implementation of Distributed IS Software Caches (D-ISSC), which take advantage of special, and temporal data localities while maintaining the capability of latency tolerance of the distributed IS memory system (D-IS). We show that D-IS and D-ISSC facilitate programming by relaxing synchronization issues. Our experimental evaluation indicates that the traffic in the interconnection network can be also significantly reduced by partial evaluation of local and remote memory accesses, and speedup of regular and irregular applications can be increased.

1. Introduction

Message passing models have emerged as expressive, efficient and well-understood paradigms for parallel programming. The proliferation of MPPs and clusters has significantly increased their popularity. Nowadays, MPI is considered a standard interface for distributed applications. It provides portability, efficiency, overlapping of communication with useful computation and many other benefits for scalable parallel computing. However, the performance of MPI is clearly bounded by the implementation of the communication interface, although a well-organized interface does not necessarily guarantee higher performance [17].

A novel technique of optimization at the communication level has been proposed and described in [8]. Non-strictness of information processing is considered on three levels: data structures, languages and execution models. In [9], the performance of Fast Fourier transform (FFT) algorithm implementations with Distributed I-Structure memory system (D-IS) and Distributed I-Structure Software Cache (D-ISSC) has been presented.

In this paper, we use the D-IS to demonstrate the power of non-strict evaluation in applications executed on distributed architectures, to hide synchronization details from programmer, and to lift the main restriction that usually strict information processing implies: complete production of data before its consumption. The D-ISSC is used to take advantage of special and temporal data localities to reduce a number of messages.

Two programs with different characteristics are compared: dense matrix multiplication (MM) and Fast Fourier Transform FFT. MM has excellent temporal locality: two input matrices are constantly referenced during the computation. On the contrary, FFT has no significant temporal data locality and reuse of data; hence, only spatial locality can be exploited.

We also use partial evaluation of local and remote memory accesses not only to remove much of the excess overhead of message passing, but also to reduce the number of messages when some information about the input or part of the input is known.

The paper is organized as follows; in the next section we describe different approaches to the MPI optimization. In section 3, we present the design of our memory system based on I-Structures. Some experimental results are presented in section 4. Lastly, conclusions are discussed.

2. MPI optimization

One of the goals of the MPI Forum is to insure that the MPI standard will continue achieving high performance as well as portability. However, the application area of MPI has been somewhat restricted to regular, coarse grained, and computation intensive applications due to

Proceedings of the Fourth ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing (SNPD'03), International Association for Computer and Information Science ACIS, pp. 277-284, 2003

the high cost of inter-process communication. In general, neither MPI-1 [22], nor MPI-2 [12] provide the best library-level way to present canonical message-passing primitives for high performance in a number of situations as shown in [30]. Several proposals were made to improve the performance of MPI.

2.1. Hardware oriented optimizations

In [30], several classes of deficiencies of the MPI standard and their solutions are presented. The modified version of MPI, is called "*Fast MPI*". *Fast MPI* utilizes message passing "temporal locality" in the specification of middleware behavior, services between the application and the middleware, and performance enhancing primitives for MPI-1. Also, *Fast MPI* uses a scheme for retaining portability.

Scali's MPI implementation [29] is another optimization applied to MPI standard over Scalable Coherent Interface (SCI) systems [18]. SCI is an ANSI and IEEE standard [6] for high-speed interconnected systems based on shared memory, with very low latency communication. *ScaMPI* is designed to support scalable systems and provides extremely low latency for short messages; furthermore, it supports multi-threaded applications and also includes a highly optimized implementation of the collective operations in MPI

2.2. Software oriented optimizations

Another optimization technique at the communication level named Active Messages (AMs), has been proposed in [11]. AMs are a low-latency mechanism for multi-processors that emphasizes the overlap of communication and computation [11]. Under AMs, the network is viewed as a pipeline operating at a rate determined by the communication overhead and with a latency, which is directly related to the message length and the network depth. An optimization of MPI, called *MPI-BIP*, is presented in [13, 16]. *MPI-BIP* is a high performance implementation of MPI for clusters connected with Myrinet using the BIP protocols. BIP (Basic Interface for Parallelism) is a low level layer for the Myrinet network [27]. As BIP supplies very limited functionality, it is not meant to be used directly by the parallel application programmer. Instead, higher layers (such as MPI) are expected to provide a development environment with a good functionality/performance ratio. MPI-BIP is based on the work done by Myricom on the MPI-GM implementation [23].

Optimizations of the MPI barrier operation are discussed in [24]. In this paper, a fast tree-based barrier synchronization scheme for 2-D meshes is proposed. The number of messages is reduced by combining the synchronization messages.

In [4], a design and implementation of the MPI collective communication instructions optimized for clusters of workstations is presented. The system consists of two main components: the MPI-CCL layer that includes the collective communication functionality of MPI and a User-Level Reliable Transport Protocol (URTP) that works as an interface with the LAN Data-Link Layer.

In [17], *MPI-StarT* for a cluster of SMPs interconnected by a high-performance network is presented, and performance and correctness issues are also discussed. The correctness issues range from the handling of arbitrarily large message sizes to deadlock-free support of non-blocking MPI operations. Performance optimizations include a shared-memory-based transport mechanism for intra-SMP communication and a broadcast mechanism that is aware of the performance difference between intra-SMP and the slower inter-SMP communication.

The most mentioned optimization techniques improve MPI performance by reducing the message management overhead, providing better hardware, or reducing the number of messages for the specific architecture. Another approach is to reduce the number of messages exploiting data structures features or static data available. The number of messages can be reduced by sending/receiving more than one data item in a single message (*spatial data locality exploitation*), the use of collective operations, or by allocating a user-level buffer to keep remote information locally in order to re-use data (*temporal data locality exploitation*). However, these optimizations are difficult to implement when elements are produced irregularly or randomly. For instance, in producer-consumer applications, a consumer can request a block of elements but a producer must wait before sending until all elements are produced or it must send data element by element; in both cases, the implementation needs to be done manually, complicating programming and reducing program performance.

2.3. Optimizations by partial evaluation

In [8], non-strictness of information processing is considered on three levels: data structures, languages, and execution models. The strictness refers to the need to compute x before $f(x)$, where x can be interpreted as operands, arguments of a function, or even input data of a program, and $f(x)$ can be interpreted as an operation, function or entire program.

Partial evaluation (PE) is an example of incomplete information processing at the level of the execution model. It allows the partial execution of a program when only some of its input data are available and generates an often faster residual program that contains operations, which are dependent on the unknown parameters. Such a

reduction of operations is a source of a program optimization.

The idea of using partial evaluation techniques for distributed applications has been considered in the recent past. Ogawa *et al*. [25] introduced a modified version of MPI called OMPI (Optimizing MPI) that removes much of the excess overhead by employing partial evaluation techniques and exploiting static information of MPI calls. OMPI reduces overhead by eliminating the error-checking and parameter checking for dynamic uses and also message buffers could be reused. Template functions are also utilized for further optimization. In [14], PE was incorporated into a library of general-purpose mathematical algorithms in order to allow the automatic generation of fast, special-purpose programs from generic algorithms. The results show speedup factors between 1.83 and 5.05 for the Fast Fourier Transform algorithm, if the size N of the input is available, where N ranges from 16 and 512. Good speedup for larger N is achieved despite the growth in code size which is $O(N\log_2 N)$.

3. Optimization of MPI programs by using non-strict information processing

In this section, we present the design of our memory system based on I-Structures called Distributed I-Structure (D-IS), and its caching mechanism called Distributed I-Structure Software Cache (D-ISSC), for distributed memory systems. D-IS and D-ISSC hide synchronization details from the programmer and reduce the communication latency by caching the split-phase transactions, so the system would combine the benefits of latency tolerance and latency reduction. An application of partial evaluation to D-IS and D-ISSC to get advantage of static information is also discussed.

3.1. Distributed I-Structure memory system

D-IS stores all information in IS elements. ISs [3, 21] are single assignment data structures where multiple updates of a data element are not permitted. Once a data element is defined, it will never be updated again. Cache coherence is already embedded. In ISs each element maintains a presence bit that has three possible states: *full, empty* or *deferred*, indicating whether valid data have been stored in the element. ISs allow the description and use of partially defined information. ISs facilitate the discovery of parallelism while timing sequences and determinacy issues would otherwise complicate its detection, and regain flexibility without losing determinacy. ISs provide non-strict data access and fully asynchronous operations.

ISs have been also implemented in the T-NG system [1, 20]. T-NG is a PowerPC 620-based multiprocessor system supporting both user-level message passing and cache-coherent shared memory. Fine-grain synchronization is provided by incorporating IS and M-Structures [5]. In [5] they combined them together for more efficient execution.

Another implementation of ISs for the Monsoon dataflow machine can be found in [1]. The global memory in Monsoon is supported in the form of IS boards that performs necessary manipulations with presence bits, including the "deferred list" management [7].

ISs have been also implemented in distributed memory systems such as the EM-X parallel system. In [31], a design and implementation of the point-to-point communication of MPI-EMX for EM-X is presented. This design includes message matching by both sender and receiver with consistency. It uses special communication supports, such as remote memory write, remote thread invocation and ISs.

3.2. Distributed I-Structure Software Cache

While the IS memory system brings many advantages, data locality is not exploited and this becomes its major drawback [21]. Therefore, an I-Structure Cache System, which caches these split-phase transactions, is required to further reduce communication latencies as well as the network load.

The idea of caching ISs is not new: in 1994, the caching of single-assignment data structures elements [10] for Abstract Shared-Memory computer was proposed. ISs support the synchronizing memory operations. The size of the cache line is a single IS element. Therefore, only temporal data locality is exploited.

In [19], a cache mechanism for multithreaded dataflow architectures is presented. It includes IS-Cache, a hardware supported cache system, to exploit data locality of shared data structures in the multiprocessor environment. The IS-Cache keeps not only the IS elements requested (I-fetch operations) by the processor but also the IS elements produced (I-store operations) by the processor.

In [26], scalable methods to deal with the storage of the deferred requests in ISs are presented. As the number of pending requests grows, there may be not enough space to store all the pending requests in the IS. This may cause a hot spot problem on the network. A distributed mechanism for the storage of the pending requests is proposed.

In [28], the design of a storage hierarchy in multithreaded architectures is presented. This architecture exploits the locality of the frame storage on the Pebbles non-blocking multithreaded model. The IS cache exploits the data locality and, hence, reduces the average turn around time of the remote requests.

A similar work is presented in [15]. A multithreaded architecture SMALL capable to exploit both coarse-grain and fine-grain instruction level parallelism is proposed. It includes a distributed data structure cache organization (DS-Cache) to reduce both network traffic and the latency in accessing remote locations making the system scalable. Coherence in caches is maintained by using a special scheme for ISs and software cache coherence mechanism for regular structures.

Another caching mechanism of IS operations implemented for multithreaded EARTH systems is presented in [21]. It was shown that ISSC increases the system utilization and improves the overall system performance by a factor of up to 95%. The cache in the advance scheme of ISSC also provides adaptability to the unpredictable communication characteristics in the system. This makes ISSC achieve the same performance without being affected by variations in the communication latency [2]. Our D-ISSC is a further development of the ISSC system. The D-ISSC works as an interface between user applications and network interface. A remote memory request is sent out to the remote host (process) only if the requested data is not available on the software cache of the local host (process).

D-ISSC is organized as sets of blocks. Each cache block (CB) contains IS elements. A cache entry is searched by using a tag associated to each CB. If data are present in the D-ISSC, the requested information is taken from the cache, otherwise, MPI message is sent to the owner of the IS. Four cases are considered:

Local access. No message is sent and the request is transferred to the local D-IS memory system.

Cache hit (value of the remote element is already in the cache). D-ISSC sends no message and the value is taken from cache.

Cache miss (value of the remote element is not in the cache). D-ISSC allocates memory for the queue of deferred reads, stores the continuation vector and sends a message to the owner of the element.

Cache miss deferred (remote element is in the cache in the *deferred state*). D-ISSC stores the continuation vector in the queue.

The most relevant features of the D-ISSC can be summarized as follows:

Deferred read operations are satisfied as soon as the element is in the "full" state. This policy guarantees the service of deferred reads in I-Structures. It maintains the spirit of producer-consumer type of synchronizations. It also avoids deadlocks.

A cache block is allocated when a cache miss occurs. CBs are allocated when a cache miss is detected. Second access to the data elements in the same CB may be issued while the first request is not satisfied. Presence bits provide a second-level data synchronization, so memory operations in the D-ISSC remain fully asynchronous. *Deferred read queues are managed dynamically.* The space to store the deferred reads can be dynamically allocated. The length of the queue for each element is bounded by the number of processes so completion of the deferred reads is not expected to be a hot spot problem. *ISs are referred by virtual addresses.* Though the single assignment property of ISs simplifies the cache coherence, some problems can occur when the IS memory space is de-allocated and re-utilized. To avoid them, logical addresses based on the unique data structure ID are used in each process.

The cache memory is organized as a hash table. Software design a cache entry is searched in the hash table using the tag field. As in a fully associative cache, CB could be mapped to any cache line.

3.3. Partial evaluation of distributed applications

The theoretical and application results obtained in the partial evaluation field are mostly associated with the fundamental properties of *sequential computations.* In [8, 9], it is shown that they can also be included in *parallel computations* in a natural way.

Split-phase transactions (*send-a-request*, *receive-a-request*, *send-a-value* and *receive-a-value*), together with the ability of defer-reads of IS elements allow partial program evaluation with ISs without losing determinacy. A *send-a-request* transaction specifies the information being requested and the process (memory) that owns the IS element and executes an MPI_Send instruction. In a *receive-a-request* transaction, the owner of the IS executes an MPI_Receive instruction. It also processes the request, checks the status of the IS element, and sends a value to the requester by an MPI_Send. If data are not present at that moment, there are two options: defer each requested element or the entire block.

If each IS element is set to the deferred state, the information about the block request can be deleted. When data are written to the elements, each deferred request is satisfied without waiting for the availability of other elements.

If the entire requested block is deferred, then, only when all IS elements are available, the block request is satisfied, and a single *send-a-value* transaction is executed by MPI_Send.

A *receive-a-value* transaction executes an MPI_Receive and writes the value to the local memory of a requester. *Send-a-request* and *receive-a-request* transactions can be completed if static information about the size of problem, number of PEs, data type, and block size is available. In that case, the programs can be partially evaluated even if the data bindings of the input are not performed. In the residual program, only *send-a-value* and *receive-a-value* transactions are left.

4. Experimental results

Experimental results are presented for a PC cluster with 4 nodes, 8 processors Pentium III in a point-to-point interconnection by 10/100 Fast Ethernet, and 512 MB of memory in each node.

As benchmarks, we use the conventional dense matrix multiplication (MM), and the Fast Fourier Transform (FFT) algorithms, both with well-known characteristics when executed on multiple processors. In the MM benchmark, each double precision 128x128 matrix is implemented as an IS. The MM has excellent temporal locality: two input matrices are constantly referenced during the computation. Conversely, FFT has no significant temporal data locality, and reuse of data, only spatial locality can be exploited.

When IS elements are not available, the MM benchmark defers each IS element separately. When elements become available each element is sent back to the requester. The FFT benchmark keeps the information about block requests and no MPI message is sent until all IS elements in that block request are fully defined and available.

Problem sizes are chosen sufficiently small so that they can easily fit into a single PC, and yet display a speed up on a multiprocessor. Indeed, the larger a problem, the better the performance improvements. This is particularly true when the memory requirements of the application cannot be satisfied in a single processor machine but can be distributed across multiple processors. Further, increasing the size of the problem transforms the application into a computationally intensive application, which has obvious parallelization advantages. Our objective here is to demonstrate that our approach will yield speed-ups, even for small problems.

Three different MPI implementations have been compared:

DIS. The program uses D-IS system without cache support.

DISSC. The D-ISSC system is used.

DISSC-Residual. In the *DISSC-Residual* program we estimate the impact of partial evaluation techniques for optimization of the program with the D-ISSC memory system. The program differs from the original *DISSC* program in that it does not include all matrix element requests (local and remote) that can be performed by partial evaluation that puts deferred read requests to the corresponding IS elements. Hence, *send-a-request* and *receive-a-request* operations are not included in the residual program. The residual program only binds IS elements and completes deferred reads. Hence, the program executes only *send-a-value* and *receive-a-value* operations.

4.1. Reduction of number of messages using D-ISSC system

Table 1 shows the number of messages for the MM and FFT benchmark programs varying the number of processor elements (PE) and cache block sizes. In this table, we see the impact of D-ISSC memory system on the messages reduction.

In the MM, D-ISSC system reduces the number of messages, when we compare with the number of messages sent by *DIS* program. For instance, for CB=1 and PEs=2, the factor of the reduction is 64, but if we increase the cache block size to eight, for the same number of processing elements the factor reaches 512.

On the other hand, for the FFT, D-ISSC has a lower impact. For example, with CB=1 there is no advantage for any number of PEs. This is caused by the no reuse of information in the FFT algorithm; hence the temporal data locality cannot be exploited.

Table 1. Number of messages of DIS and DISSC programs for the MM and FFT benchmarks varying the number of processing elements (PE) and cache block sizes (CB).

Program		Number of messages		
		2 PEs	4 PEs	8 PEs
MM				
	DIS	2,097,152	3,145,728	3,670,016
	DISSC CB=1	32,768	98,304	229,376
	DISSC CB=2	16,384	49,152	114,688
	DISSC CB=4	8,192	24,576	57,344
	DISSC CB=8	4,096	12,288	28,672
FFT				
	DIS	16,384	40,960	81,920
	DISSC CB=1	16,384	40,960	81,920
	DISSC CB=2	8,192	20,480	40,960
	DISSC CB=4	4,096	10,240	20,480
	DISSC CB=8	2,048	5,120	10,240

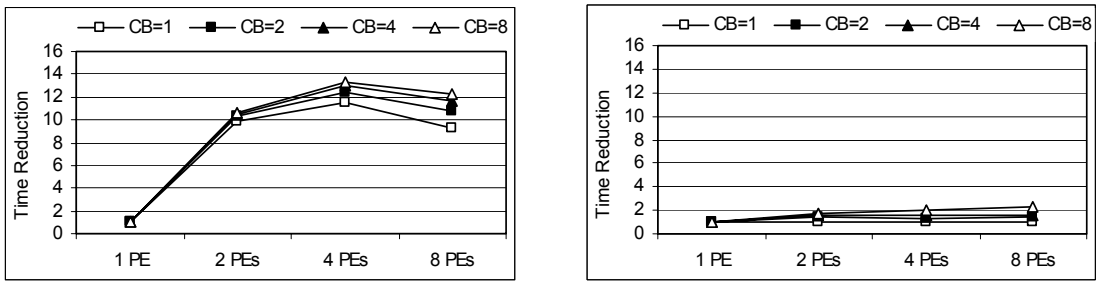

Figure 1. Ratio of D-ISSC over D-IS, for MM (a) and FFT (b) benchmark programs, for different number of processors and cache block sizes.

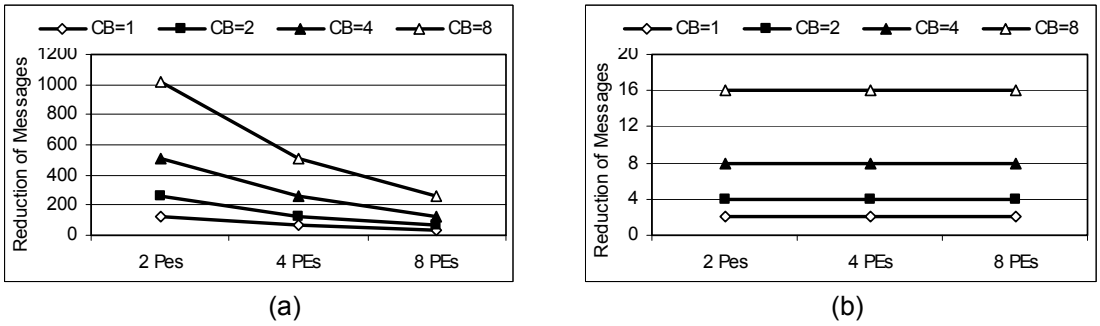

Figure 2. Ratio between number of messages sent by the D-IS program over number of messages sent by the D-ISSC Residual programs for the MM (a) and FFT (b) benchmarks varying the number of processes and cache block sizes.

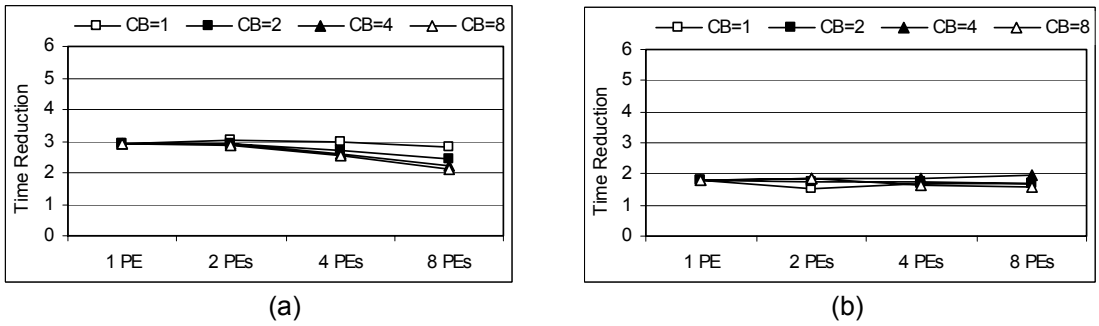

Figure 3. Time reduction obtained when applying PE technique to DISSC programs for the MM (a) and FFT (b) benchmarks varying the number of processes and cache block sizes

However, if the cache block size is increased, then we get an advantage of the spatial data locality. For PEs=2, the number of messages is reduced by factor of two for CB=2 and eight for CB=8.

4.2. Time reduction using D-IS and D-ISSC

In the previous section, we show the message reduction by D-ISSC memory system. Now we demonstrate how this reduction impacts the execution time. Figure 1 shows how much faster distributed programs with D-ISSC mechanism are than those without the cache support for different number of processors. Significant reduction of the number of messages in MM with D-ISSC makes it more than eight times faster (Figure 1a) than corresponding IS program. Less data locality exploitation in the FFT programs results in a lower speedup of the programs with D-ISSC. FFT with D-ISSC is at least 3% slower than without D-ISSC. This

is caused by the D-ISSC mechanism overhead. However, if the cache block size is increased, for instance, to eight, for PEs=8, the program is 2.3 times faster (Figure 1b).

4.3. Reduction of a number of messages using D-ISSC system and partial evaluation technique

In this section, partial evaluation is applied to the *DISSC* programs for further reduction of messages. If the problem size (the size of matrices for MM and the input nodes for FFT), the number of PEs, data type, block size, and the MPI communicator are known, all remote requests can be executed even the data are not available. Hence, the specialized residual programs do have a half of the split phase transactions.

Figure 2 shows the impact of D-ISSC and partial evaluation together on the reduction of messages when varying number of PEs and cache block sizes. In Figure 2a, we see that the ratio of the number of messages sent by the MM D-IS program over the number of messages sent by the MM D-ISSC is 1024 for CB=8. It is decreased when adding more PEs, or when the cache block size is decreased.

This is caused by the difference in exploiting temporal and spatial data locality in original program and residual one. In the residual program, all IS requests sent by one message are deferred separately, which causes an increase in the number of messages in the residual program.

The ratio is a constant regarding the number of PEs for FFT program (Figure 2b), where the D-ISSC memory system keeps and defers the entire requested blocks.

If we consider the number of messages sent by *D-IS* programs as 100% then we can conclude that more than 95% of the messages in the MM benchmark program and more than 75% in the FFT program can be reduced using the D-ISSC system. This considerable reduction decreases the traffic in the interconnection network and improves the overall system performance.

4.4. Time reduction using partial evaluation technique.

In Figure 3, we show the acceleration of *D-ISSC* residual programs as compared to the *D-ISSC* original ones. Acceleration is presented for different number of PEs and cache block sizes.

It shows that *D-ISSC-Residual* programs for the MM algorithm are between 2 and 3 times faster them corresponding *D-ISSC* program, while *DISSC-Residual* for FFT is 1.5-1.8 times faster. In both cases the time reduction is mainly caused by the reduction of the number of messages. As mentioned in section IV.B residual programs send twice fewer messages than the original programs. Also, we see that the increment of cache block size does not significantly change the execution time of residual programs.

5. Conclusions

In this paper, we have presented an approach to optimize MPI program by their non-strict evaluation. We have described our D-IS memory system that based on incomplete data structures, where IS fetch request may arrive at an IS element before the corresponding IS store completes. Whether an I-fetch is deferred or not depends upon scheduling, and ultimately on algorithm and machine configuration, but not upon synchronization issues.

Moreover, the split-phase memory access schemes of MPI and D-IS allow overlap long communication latencies with useful computations. D-IS allows performing deterministic asynchronous non-strict memory accesses which simplify the programming of distributed programs. We have also presented a distributed software cache mechanism to reduce the communication latency by caching the split-phase transactions, so that the system would combine the benefits of latency tolerance and latency reduction.

We have shown that the positive impact is more significant for systems with large latency such as PC clusters with commodity NICs. By exploiting the temporal and spatial data localities, the number of messages decreases making programs more scalable, and avoids the saturation of interconnection network leaving the resources free for other applications. The use of D-IS and D-ISSC has a cost in time, but by taking advantage of the data locality, the D-ISSC exceeds this cost and improves the system performance.

On the other hand, we have shown that split-phase operations and deferred read properties of the D-IS make the concept of partial evaluation of distributed programs feasible. Partial evaluation allows a reduction in the number of messages in data-independent applications. It can also be applied for further program optimizations by constant propagation, loop unrolling, and polyvariant specialization in order to get fully advantage of static information available.

Acknowledgments. The authors take pleasure in acknowledging Ana Melisa Heras Luzanilla and Jazmin Romero Gonzalez for their participation in the experimental analysis. This work is partly supported by CONACYT (Consejo Nacional de Ciencia y Tecnología de México) under grant #32989-A and by the National Science Foundation under Grants No. CSA-0073527 and INT-9815742. Any opinions, findings, conclusions, or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation or of CONACYT.

6. References

[1] B. Ang, Arvind, and D. Chiou "StarT the next generation: Integrating global caches and dataflow architecture". In G. Gao, L. Bic, and J.-L. Gaudiot, Ed. *Advanced Topics in Dataflow Computing and Multithreading*, pp. 19-54. IEEE, 1995.

[2] J-N Amaral, W-Y Lin, J-L.Gaudiot "Exploiting Locality in Single Assignment Data Structures Updated Through Split-Phase Transactions" *Cluster Computing Journal* Vol. 4 No. 4 pp. 281-293 2001.

[3] Arvind, R-S Nikhil, K-K. Pingali "I-Structures: Data Structures for Parallel Computing." *ACM Transaction on Prog. Languages and Systems*, Vol. 11 No. 4 pp. 598-632. 1989.

[4] J.Bruck, D. Dolev, C-T Ho, M-C Roşu, R. Strong. "Efficient Message Passing Interface (MPI) for Parallel Computing on Clusters of Workstations" *Journal of Parallel and Distributed Computing*, Vol. 40 No. 1 pp. 19-34. 1997.

[5] P-S Barth, R-S Nikhil, Arvind. "M-Structures: Extending a Parallel, Non-strict Computation Structures". *Proceedings on Functional Programming and Computer Architecture*, Cambridge, MA, August pp. 28-30, 1991

[6] Ballot Review Committee of the IEEE Microprocessor Standards Committee, 'SCI - Scalable Coherent Interface' P1596/D2.00 18Nov91.

[7] D. Culler, G. Papadopoulos. "The Explicit Token Store" *Journal of Parallel and Distributed Computing*, Vol. 10, No. 4, 1990, pp. 289-308.

[8] A. Cristobal, A.Tchernykh, J-L. Gaudiot, W-Y Lin. "Non-Strict Execution in Parallel and Distributed Computing", *International Journal of Parallel Programming*, Kluwer Academic Publishers, New York, vol. 31, 2, p. 77-105, 2003

[9] A. Cristobal, A. Tchernykh, J-L. Gaudiot. "Non-strict Evaluation of the FFT Algorithm in Distributed Memory Systems", *EuroPVM/MPI* 2003, Lectures Notes in Computer Science, Springer-Verlag, 2003 (to appear)

[10] J. Dennis, G. Gao. "On memory models and cache management for shared-memory multiprocessors." *CSG MEMO* 363, LCC-MIT. March 1995.

[11] T. Eicken, D. Culler, S. Goldstein, K. Schauser. "Active Messages: a Mechnisim for Integrated Communication and Computation." In Proceedings of the *19th International Symposium on Computer Architecture*. pp. 256-266, May 1992.

[12] A. Geist, W. Gropp, S. Huss-Lederman, S. Lumsdaine, E. Lusk, W. Saphir, A. Skjellum, M. Snir. "MPI-2: Extending the Message-Passing Interface." In L. Bouge, P. Fraigniaud, A. Mignotte, and Y. Robert, editors, LNCS, 1/1123, pp. 128-35. Spring Verlag*, Euro-Par '96* Parallel Processing. 1996.

[13] W. Gropp, E. Lusk, N. Doss, A. Skjellum. "A high-performance, portable implementation of the MPI message passing interface standard." *Parallel Computing*, Vol 22 No 6 pp.789–828, Sep. 1996.

[14] R. Gluck, R. Nakashige, R. Zochling. "Binding-time analysis applied to mathematical algorithms." In J. Dolezal and J. Fidler, editors, *17th IFIP Conference on System Modelling and Optimization*. 1995.

[15] R. Govindarajan, S. Nemawarkar, P. LeNir. Design and performance evaluation of a multithreaded architecture. *The first international symposium on High-Performance Computer Architecture*, Raliegh, pp. 298-307. 1995.

[16] P. Geoffray, L. Prylli, B. Tourancheau. "BIP-SMP: High Performance Message Passing over a Cluster of Commodity SMPs." *Super Computing* 99, Portland, USA. 1999.

[17] H. Parry, J-C. Hoe, "MPI-StarT: Delivering Network Performance to Numerical Applications", *The 1998 ACM/IEEE conference on Supercomputing* pp. 1-15. Nov. 1998

[18] D. James, A. Laundrie, S. Gjessing, G-S. Sohi, 'Scalable Coherent Interface", *IEEE Computer* Vol. 23, No 6 pp. 74-77. 1990.

[19] K. Kavi, A. Hurson, P. Patadia, E. Abraham, P .Shanmugam, "Design of cache memories for multithreaded dataflow architecture", *ISCA 95*, pp. 253-264. 1995.

[20] D. Kranz, B-H. Lim, A. Agarwal, D. Yeung, "Low-cost support for fine grain synchronization in multiprocessors.", Kluwer Academic publishers, 1994.

[21] W-Y. Lin, J-L. Gaudiot, "The design of an I-Structure Software Cache System", In Proc. *MTEAC* 1998.

[22] MPI Forum. MPI: A Message Passing Interface. 1993. http://www.mpi-forum.org/docs/docs.html

[23] Myricom. The GM API, December 1998. http://www.myri.com/GM/doc/gm toc.html.

[24] S. Moh, C. Yu, B. Lee, H-Y. Youn, D. Han, D. Lee, "4-ary Tree-Based Barrier Synchronization for 2-D Meshes without Nonmember Involvement", *IEEE Transactions on Computers*, Vol. 50, No. 8. 2001.

[25] H. Ogawa, S. Matsuoka, "OMPI: Optimizing MPI programs using Partial Evaluation", *IEEE/ACM Supercomputing Conference*, Pittsburgh, November 1996.

[26] G-M. Papadopoulos. "Implementation of a General-purpose Dataflow multiprocessor", PhD thesis, Laboratory for Computer Science, MIT, August 1988.

[27] L. Prylli, B. Tourancheau, "BIP: a new protocol designed for high performance networking on myrinet", In *Workshop PC-NOW, IPPS/SPDP98*, Orlando, USA, 1998.

[28] L. Roh, W-A. Najjar. Design of storage hierarchy in multithreaded architectures. In *IEEE proceedings of MICRO-28*, pp. 271-278. 1995.

[29] SCALI. 2001.http://www.scali.com.

[30] A. Skjellum, "High Performance MPI: Extending the Message Passing Interface for Higher Performance and Higher Predictability", *PDPTA'98*, Las Vegas. July 1998.

[31] K-B. Theobald, J-N. Amaral, G. Herber, O. Maquelin, X. Tang, G-R. Gao, "Overview of the Threaded-C Language", *Technical Memo* 19, CAPSL, University of Delaware, March 16, 1998.