Partial Evaluation Technique
for Distributed Image Processing

A. Tchernykh[a], A. Cristóbal-Salas[b], V. Kober[a], and I. A. Ovseevich[c]

[a] Computer Science Department, CICESE Research Center Ensenada, BC, Mexico 22830
[b] School of Chemistry Sciences and Engineering, University of Baja California, Tijuana, B.C. Mexico 22390
[c] Institute for Information Transmissions Problems, Russian Academy of Sciences, Bol'shoi Karetnyi 19, Moscow, Russia
e-mail: chernykh@cicese.mx; cristobal@uabc.mx; vitally@iitp.ru; ovseev@iitp.ru

Abstract—In this paper, a partial evaluation technique to reduce communication costs of distributed image processing is presented. It combines application of incomplete structures and partial evaluation together with classical program optimization such as constant-propagation, loop unrolling and dead-code elimination. Through a detailed performance analysis, we establish conditions under which the technique is beneficial.

1. INTRODUCTION

Typical image processing tasks require a large amount of processing power, larger than can be achieved by current state-of-the-art workstations. Parallel processing using distributed systems appears to be the only solution to obtain sufficient processing power for handling all image processing levels. In spite of the fact that the exchange of information remains a critical bottleneck in such applications with inherent high communication costs, the message passing paradigm has significantly increased in popularity with the proliferation of clusters and GRID technology. It appears that the optimization of parallel programs is as equally demanding as the design of a parallel algorithm itself. Communication cost depends on many factors such as memory manipulation overheads (message preparation, message interpretation), network communication delays, etc. Reducing this cost is vital to achieve good performance. There are several strategies to minimize it, for instance, by computation and communication overlapping, network topology optimization, bandwidth increasing, reduction of number of messages (message coalescing, caching messages), messages pipelining, etc.

High performance of the fast Fourier transform (FFT) and wavelet transforms is a key issue in many image applications [1]. There have been several attempts to parallelize and optimize FFT. For example, an algorithm suitable for 64-processor nCUBE 3200 hypercube multicomputer was presented in [2] where a speedup of up to 16 with 64 processors was demonstrated. The FFT implementations for hypercube multicomputers and vector multiprocessors were discussed in [3]. A parallel FFT algorithm for 64-processor Intel iPSC was described in [4]. Techniques for parallelization of the multidimensional hypercomplex FFT are considered in [5]. The FFT implementations based on incomplete data structures were presented in [6]. Speedup factors between 1.83 and 5.05 were archived for shared memory computers if the size of the input N is available. Good speedup is achieved despite the significant growth ($N_{\log 2}N$) of the code size.

Nevertheless, in spite of the reduction in complexity and time, FFT and wavelet transforms remain expensive mainly for distributed memory parallel computers where network latency significantly affects the performance of the algorithm. In most cases, the speedup gained by parallelization is limited due to inter-process communication. That is why programming for distributed architectures has been somewhat restricted to regular, coarse-grained, and computation-intensive applications. FFT exploits fine grain parallelism, which means that an improvement at the communication level plays an extremely important role.

The aim of this work is to demonstrate that the performance benefits of incomplete information processing and partial evaluation can be brought to distributed image processing in a high level manner that is transparent to users. To this end, partial evaluation is used not only to remove much of the excess overhead of sequential and shared memory parallel computation, but also to distributed application to reduce the number of messages when some information about the image or part of the image is known. The work demonstrates that good parallel speedups are attainable using MPI and can be integrated into existing distributed environment. 1D-Fast Fourier and 2D Haar wavelet transforms' optimization is presented.

The rest of the paper is organized as follows. In Section 2 a general description of the proposed optimization technique is presented. Partial evaluation, incomplete data structures and incomplete data structures

software cache are described. Experimental results are presented in Section 3. Partial evaluation of distributed FFT and 2D Haar wavelet transform is discussed. Finally, conclusions are described.

2. PROPOSED OPTIMIZATION TECHNIQUE

Theoretical and application results obtained in the partial evaluation field are mostly associated with the fundamental properties of sequential computations. In this paper, these results are generalized and included to distributed computations in a natural way. Distributed incomplete data structures and distributed partial evaluation technique are designed and studied for distributed image processing optimization.

2.1. Partial Evaluation

Partial evaluation [7, 8] is an automatic program transformation technique which allows partial execution of a program by pre-computing parts of the program that depends on known input parameter settings, producing a residual program that depends on the rest of the inputs. It enables the construction of specialized program from general highly parameterized software systems. Specialization turns a general program into an efficient one, optimized for a specific parameter setting. It is similar in concept to, but in several ways stronger than, a highly optimizing compiler. It can take into account run-time information. Specialization can be done automatically or with human intervention. Partial evaluation may be considered a generalization of the conventional evaluation [9].

Adopting notations of Mogensen and Sestoft [8], p is used for the program text, and $[[p]]$ for the function computed by p. Let p require two inputs, x_1 and x_2. When the specific values d_1 and d_2 are given for the two inputs, the program can be executed producing a result. Let us denote by $[[p]]_L[d_1, d_2]$ the result of running p with input values d_1 and d_2 on an L-machine. When only one input value d_1 is given, p cannot be executed, but can be partially evaluated producing a version p_{d1} of p specialized for the case $x_1 = d_1$. A partial evaluator is a program peval, which performs partial evaluation. Hence, given a program p and d_1, it produces a residual program p_{d1}, with one input such that $[[peval]]_L [p, d_1] = p_{d1}$. It satisfies $[[p_{d1}]]_T[d_2] = [[p]]_S[d_1, d_2]$ for all d_2 or $[[peval]]_L[p, d_1]]]_T[d_2] = [[p]]_S[d_1, d_2]$, where L is the language in which peval is written, S is the language of the source program, and T is the language of the target program.

Although simple in concept, partial evaluation has been used in such areas as compilation, scientific computing, meta-programming, computer graphic, pattern matching, and others [10–17].

It has been shown [18, 19] that the majority of scientific programs and, in particular, image processing programs are static structured, that is, the programs'

skeleton and communication patterns can be determined if the size of the problems and number of processors are known. Thus, they are highly suitable for partial evaluation. As an example, we refer to [20] where the partial evaluation technique is incorporated in a library of general-purpose mathematical algorithms in order to allow automatic generation of fast, special-purpose programs from generic algorithms.

2.2. Partial Evaluation of Distributed Applications

The idea of using partial evaluation for distributed applications has been considered in the recent past. A partial evaluation model that distributes the work of creating the specializations to distinct computational agents called specialization servers is presented in [16]. Each specialization server can perform work on an arbitrary specialization, given its static configuration. As specialization produces more and more static configurations, it is necessary to distribute the work among the specialization servers. In [21], the OMPI (Optimizing MPI) system that removes much of the excess overhead by employing partial evaluation and exploiting static information of MPI calls is considered.

The important question now is would it be possible to use partial evaluation not only to remove the excess overhead of distributed processes as in the original approach, but also to reduce the number of messages between them? The answer is effectively yes. The problem is to assess how to partially evaluate MPI_send and MPI_receive operations. In this paper, the representation of messages in the form of split-phase transactions such as send-a-request, receive-a-request, send-a-value, and receive-a-value are proposed that allows evaluating MPI programs partially without losing determinacy. To support split-phase transactions, distributed incomplete data structures are designed (see 2.3).

Figure 1 shows the main phases of distributed programs partial evaluation. A program code and a set of known parameters are given as an input. The output is a residual (optimized) code. Two main steps are considered: pre-processing and message elimination.

At the preprocessing step, the MPI code is transformed to facilitate detection of static memory accesses. The main parallelized code is replicated in accordance with the number of processes given, constants are propagated, dead codes are eliminated, and loops are unrolled. At the message elimination step, static memory accesses are evaluated by inserting a special instruction into the corresponding processes to perform the remote request locally. After evaluation of static memory requests, at the second step, partial evaluation of local codes is performed to complete execution of requests that have enough information.

The code size and memory consumption may be increased while improving efficiency. However, the same occurs with traditional partial evaluation tech-
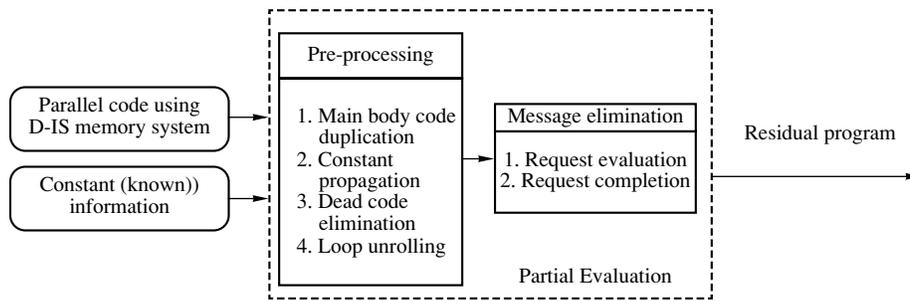
Fig. 1. General view of the optimization.

nique. The regulation of the extra code generation can be done by restriction of unfolding, recursion depth, and loop unrolling automatically or with human interaction during partial evaluation step.

Before going into details of experimental analysis, the design of Incomplete data Structures (ISs) and Distributed Incomplete data Structures (D-IS) memory systems is presented.

2.3. I-Structures

In a multiprocessor system, arbitrary (chaotic) writing and reading may occur because of parallel computations. When processors communicate, they need to synchronize to ensure that valid data are used and to avoid race conditions. If some required data are not available, the processor must either wait them (using coarse or fine grain synchronization), or switch to another thread, and occasionally poll for the arrival of the data or wait for an interrupt announcing the arrival of the data.

The distributed message passing paradigm provides many benefits for scalable parallel computations. However, one of its drawbacks is that it also allows unrestricted access to data. Its performance is bounded by the performance of the underlying communication interface. However, an efficient interface does not necessarily guarantee a high performance implementation.
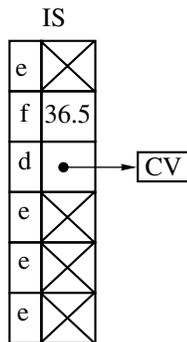

Fig. 2. IS.

In the paper, to increase message passing performance, the elimination of synchronization issues, nonstrict asynchronous data access, and the reduction of the number of messages are applied.

Incomplete data structures [22–28] provide nonstrict data access and fully asynchronous operations that makes them highly suited for the exploitation of fine-grain parallelism. A request for IS fetch may arrive at an individual IS element before the corresponding IS store completes. It facilitates asynchronous access. Values production and consumption proceed with a looser synchronization than conventionally understood. To do it, the run-time system should be capable of testing for an element state. Each element maintains a presence bit which has three possible states: full (f), empty (e), or deferred (d), indicating whether a valid data has been stored in the element (Fig. 2).

2.4. Distributed I-Structure Memory System (D-IS)

D-IS is a message passing communication library that implements the functionality of ISs on the top of MPI. Each MPI process manages a local IS memory system arranged in a linked list (Fig. 3). Remote operations are performed using split-phase transactions within MPI point-to-point routine calls. Exchange of information involves a send-request, receive-value on the requester side and receive-request, send-value on the side of the owner of the IS. Split-phase operations tolerate request latencies by decoupling initiators from receivers of communication transactions [29]. D-ISs facilitate the exploitation of parallelism while timing sequences and determinacy issues would otherwise complicate its detection, and regain flexibility without losing determinacy [28, 30, 31].

D-IS permits reading an IS element even before a value is bound to that memory location. This feature breaks the restrictions unnecessarily imposed by sequential systems, which demand the complete production of data before their consumption. The write policy is a write-through one to ensure that data will be available as soon they are produced.

D-IS is a further development of the IS memory system. As D-IS runs on the top of MPI, it has most of MPI features such as portability and efficiency on different
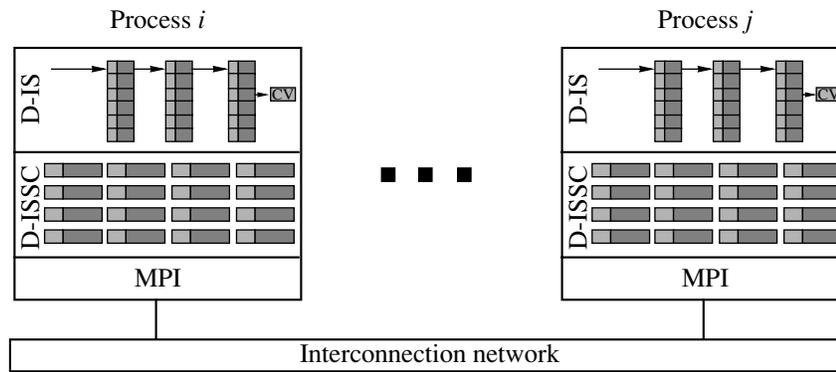
Fig. 3. D-IS and D-ISSC.

architectures. The D-IS memory system has been tested on NUMA S2MP ORIGIN 2000 supercomputer and Pentium IV cluster. However, the overhead of D-IS management becomes its major drawback [26].

Several efforts related to the optimization of IS memory systems using a caching mechanism (IS Software Cache, ISSC) have been proposed [29, 30, 32, 37].

In order to extend this mechanism for distributed address space, Distributed IS Software Cache (D-ISSC) is designed. It is a further development of the ISSC system. D-ISSC takes advantage of spatial and temporal localities without hardware support. The spatial data locality refers to the situation when, due to the long latency and unpredictable characteristics of a network, a second remote access to the data elements in the same data block (cache line) may be issued while the first request is still pending. The temporal data exploitation refers to the reuse of data which are already in the cache. Due to the inherent cache coherence feature of D-ISs, no cache coherence problem exists. This significantly reduces the overhead of the cache system functionality. D-ISSC works as an interface between user applications and network and is implemented as an extension of the MPI library. It makes the cache system portable and provides non-blocking communication facilities.

Distributed programs where parallel control structure is determined by the size of the problem (data-independent programs) as well as by the number of parallel processes can be partially evaluated even if the data bindings of the input are not performed. The MPI_Send (send-a-request) instruction can be executed because the data decomposition and data localization are available.

By using IS, the receive-a-request transaction can also be evaluated. The owner of the value executes the MPI_Receive instruction, checks the status of the corresponding element, and, if it is available, sends a value back to the requester by the MPI_Send instruction. Otherwise, it stores the request as a deferred read to this D-IS element. Hence, the residual program includes only send-a-value and received-a-value operations.

Later, when a value of the element is produced and written, the owner of the element finds out the list of pending reads (continuation vectors) and sends a value to the requestors by executing MPI_Send instructions. A receive-a-value transaction executes an MPI_Receive instruction and writes the value to the local memory of a requester. Hence, the residual program may be evaluated completely.

It is important to mention that the residual program may be executed as many times as needed with different input data. It is expected to be faster than the original program.

3. EXPERIMENTAL RESULTS

In this section, designed distributed incomplete data structures and partial evaluation technique applied to distributed applications optimization are studied. Performance evaluations of the MPI FFT and the 2D Haar wavelet transform are presented.

3.1. FFT

In this section, performance evaluation of the MPI FFT algorithm with 2048 double precision complex data on an SGI ORIGIN2000 with 8 MIPS R10000 processors running at 195MHz, 1280MB of main memory, and a network bandwidth of 800 MBs/sec is discussed. Six different implementations have been compared:

(1) FFT. It is the basic implementation with MPI.

(2) FFT-Residual. This program differs from FFT in that all send-a-request and receive-a-request transactions are performed at the partial evaluation step. Hence, they are not included in the residual program. Each IS element has a vector of deferred reads. The residual program includes operations to bind elements, complete pending requests, and execute send-a-value and receive-a-value transactions.

(3) FFT-DIS. Remote requests are managed by the D-IS memory system.

(4) FFT-DIS-Residual. A residual program differs from the original FFT-DIS program in that all requests

Number of messages varying the number of processors

MPI programs	Cache line	2 PEs	4 PEs	8 PEs
FFT		16.384	40.960	81.920
FFT-Residual		8.192	20.480	40.960
FFT-DIS		16.384	40.960	81.920
FFT-DIS Residual		8.192	20.480	40.960
FFT-DISSC	CL = 1	16.384	40.960	81.920
FFT-DISSC-Residual		8.192	20.480	40.960
FFT-DISSC	CL = 4	4.096	10.240	20.480
FFT-DISSC-Residual		2.048	5.120	10.240
FFT-DISSC	CL = 8	2.048	5.120	10.240
FFT-DISSC-Residual		1.024	2.560	5.120

for D-IS data items, local or remote, and receive-a-request operations are performed during the partial evaluation step.

(5) FFT-DISSC. The D-ISSC system is used.

(6) FFT-DISSC-Residual. Each element of the D-IS has a vector of deferred reads. The residual program only binds elements, completes pending requests, and executes send-a-value and receive-a-value transactions. To support a cache line (CL) mechanism, the vector has one extra element which counts how many elements in a requested cache line have been produced.

Table 1 shows the number of messages varying numbers of processors. It may be seen that D-ISSC does not reduce the number of messages when a single D-IS data item is cached (CL = 1). With caching 4 and 8 D-IS data items, the reduction of messages obtained by FFT-DISSC is 4 and 8, respectively. This shows that the FFT algorithm has no significant temporal data locality; hence, no re-use of data is optimized by the cache. Only spatial locality is exploited by the cache line support. Table 1 shows how D-ISSC contributes to the messages reduction. Increasing the size of the cache line proportionally decreases the number of messages, for example, the total reduction in the FFT-DISSC–
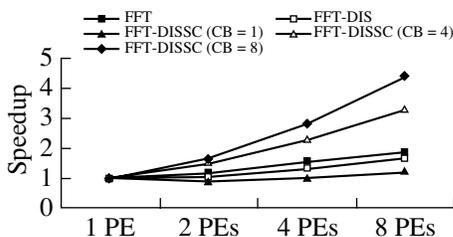
Residual (CL = 8) is 16 times comparing with the FFT. In the residual programs, the number of messages is reduced by a factor of two as compared to the original ones, irrespective of the number of PE. MPI-send messages corresponded to the data requests are eliminated. It is important to note that the messages reduction diminishes not only the program execution time, but also improves the computer system behavior by reducing the saturation of the communication system.

Figure 4 shows the speedup varying numbers of PEs. FFT-DIS has the lower speedup than FFT due to the D-ISs management overhead. FFT-DISSC with CL = 1 has a lower speedup than FFT-DIS because of extra overhead and the lack of temporal data locality. Nevertheless, the spatial data locality exploited by the D-ISSC mechanism contributes to the speedup of the FFT program. For instance, for 8 PEs, the speedups are from 1.19 to 4.39 varying CL from 1 to 8.

Figure 5 presents the relative time reduction of FFT-DIS and FFT-DISSC, with different cache line (cache block) sizes over the original FFT program, varying the number of PEs. FFT-DIS and FFT-DISSC with CL = 1 are not faster than FFT. The speedup is increased when CL = 4, and 8. For PEs = 8, FFT-DISSC has a relative time reduction of 1.46 and 2.01, respectively. The time reduction is higher than the overhead of D-ISs and D-ISSC management.

The degree of parallelizability of the residual programs is presented in Fig. 6. It shows the speedups of the residual programs varying number of PEs. It shows that the speedup of FFT-Residual, and FFT-DIS-Residual is relatively small, between 2 and 3 for 8 PEs. For the same number of PEs, the speedup of FFT-DISSC-Residual is increased from 2.1 to 5.35 varying CL from 1 to 8.

To evaluate the impact of partial evaluation on the performance, a time optimization coefficient $S_p^o = T_O/T_R$ is calculated. S_p^o is the ratio of the execution time T_O taken by the original program over the time T_R taken by the residual one. Figure 7 shows the S_p^o for the benchmark program with and without cache system versus the number of processors. FFT-Residual is 20–50% faster (depending on the number of PEs) than FFT.
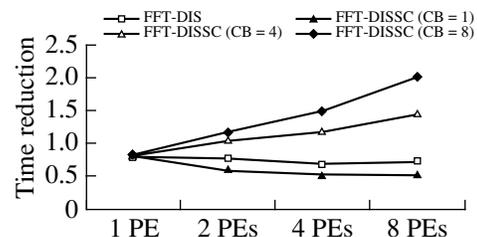

Fig. 4. Speedup of FFT, FFT-DIS and FFT-DISSC (with cache line sizes 1, 4 and 8) varying number of PEs.

Fig. 5. Time reduction of FFT-DIS and FFT-DISSC (with cache line sizes 1, 4 and 8) over FFT, varying the number of PEs.

FFT-DIS-Residual is about 70% faster than the original FFT-DIS program. The time reduction of FFT-DISSC-Residual program is slightly higher. It is about 90% when CL = 1 and 46% for CL = 8. Increasing the CL reduces the number of messages and, hence, fewer messages are removed by partial evaluation.

3.2. 2D Haar Wavelet Transform

In this section, experimental evaluation of the 2D Haar wavelet transform (2D-HWT) applied to a 1024 × 1024 image is presented. Four dual-Pentium IV PC cluster with 10/100 Fast Ethernet point-to-point inter-connection, and 512 MB of each node memory is used.

The Haar wavelet transform is the first known wavelet, proposed by Alfred Haar in 1909 [34]. It is also the simplest wavelet. As opposed to the functions sine and cosine used in Fourier transforms, a wavelet not only has locality in the frequency domain, but also in the time or spatial domain. The algorithm produces an output containing the average of the original image together with the detail information of the image. In studied implementation, collective communications, cache mechanism, message coalescing, data locality exploitation, or other program tune-ups are not used. These restrictions are set to observe how much gain can be obtained by the partial evaluation technique.

In the benchmark program, different percentage of the known input image is assumed. This assumption is reasonable in digital image processing where images may contain a constant background or fixed objects. In the experiments, the 2D-HWT with D-IS memory system is studied.

Evaluation results are presented for different percentages of static (known) image, network latencies, and number of processing elements (PEs). Let:

D-IS be the original MPI program without optimization;

D-IS(k) be the optimized program when *k* percent of the image is known. If *k* = 0, partial evaluation still can be performed because the data requests can be evaluated when the image size is provided.

Figure 8 shows the number of messages in the optimized and non-optimized programs varying the number of PEs. Comparing *D-IS* and *D-IS(0)*, we can see that half of the messages is eliminated by setting the image size and the number of processing elements. Under available information, the memory requests can be evaluated completely even without knowledge of the image. The other half of messages required the value of the pixels, so they are kept in the residual program. The number of messages is reduced when the number of PEs increased due to data locality. Comparing *D-IS(0)*, *D-IS(5)* and *D-IS(20)* in Fig. 8; it can be seen that a certain part of data requests can be answered, thereby eliminating send-a-value messages.

The reduction of the ratio between the number of messages sent by the *D-IS* program over the number of
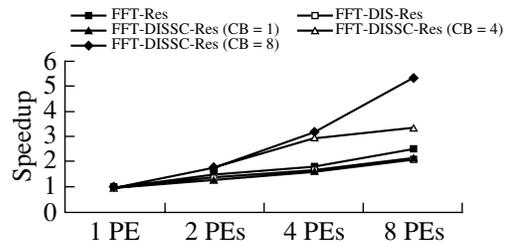

Fig. 6. Speedup of FFT-Residual, FFT-DIS-Residual, and FFT-DISSC-Residual (with cache line sizes 1, 4 and 8) for different number of PEs.
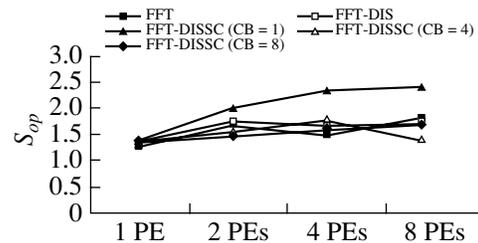

Fig. 7. S_{op} for FFT with and without cache system varying the number of PEs.
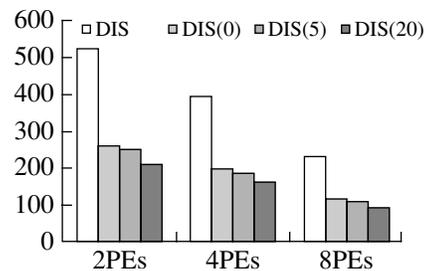

Fig. 8. Number of messages Fig. 9. Reduction in the message varying the number of PEs and the rate when part of the image k percentage of the known image.
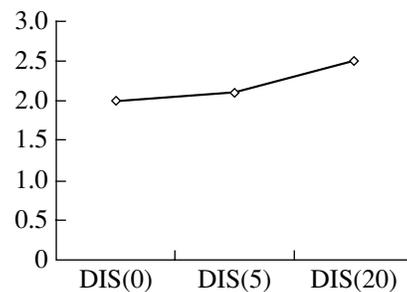

Fig. 9. Reduction in the message rate when part of the image *k* (0%, 5%, and 20%) is known.
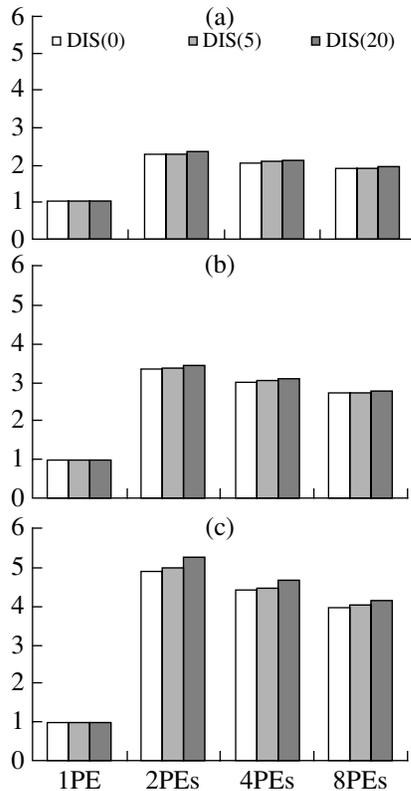
Fig. 10. Execution time reduction rate varying the number of PEs, *k* and the interconnection network speed (a) twice faster (b) original and (c) twice slower network speeds.

Fig. 11. Speedup of *D-IS, D-IS(0), D-IS(5), D-1S(20)* programs with different numbers of PEs. We present data for (a) twice faster (b) original and (c) twice slower interconnection network.

messages sent by the *D-IS(k)* programs is presented in Fig. 9. As may be seen, this ratio is at least 2 and increases when the part of the image is known. This tendency is kept for 2, 4, and 8 PEs.

Figure 10 shows the execution time reduction of the *D-IS* program varying the percentage of constant information, number of PEs, and the interconnection network latency. Execution time reduction rate is the ratio of *D-IS* execution time over *D-IS(p)* execution time. The impact of the partial evaluation with different interconnection network latencies is observed. In Fig. 10a, the interconnection network is twice faster than network in Fig. 10b and four times faster than in Fig. 10c. The reduction rate is higher when the interconnection network is slower. It means that the proposed technique makes single assignment memory system more robust and latency tolerant. The optimization gain is not observed for one PE. It proofs that the main contribution is based on the reduction of remote memory operations instead of local memory operations. An increment of the reduction ratio is observed when *k* is increased from 0 to 20 due to eliminating a part of send-a-value messages. It is small because the processing of those messages is less time-consuming compared with send-a-request and receive-a-request messages. The
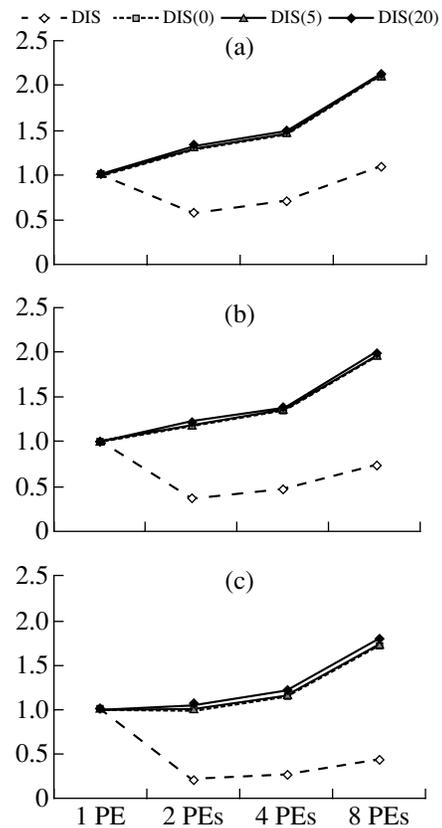
optimization gain is reduced when the number of PEs is increased. This is due to the data distribution between PEs. When more PEs are added, more messages are required to exchange information.

Figure 11 shows the speedup obtained by 2D-HWT increasing the number of PEs and varying the interconnection network speed by a factor of two. Figures 11a, 11b, and 11c show that D-IS have a speedup below one for all networks, so it is slower than its sequential counterpart. This is due to the time consuming messages exchange. However, with the introduction of more PEs, the program begins speeding up. When the interconnection network is fast enough, the speedup becomes higher than one (see Fig. 11a *D-IS* with 8 PEs). However, when the partial evaluation is applied to *D-IS(0)*, we note a positive speedup. This tendency is also valid for *D-IS(5)* and *D-IS(20)*. In these cases, the overhead introduced by the management of I-Structures and the communication times can be hided by partial evaluation producing a faster optimized code.

4. CONCLUSIONS

In this paper, we have proposed approaches to distributed image processing optimization based on

incomplete data structures and partial evaluation. Experimental results of Fast Fourier and Haar wavelet transforms are presented. It is demonstrated that the performance benefits of incomplete information processing can be brought to distributed image processing in a high level manner that is transparent to users. FFT and 2D-HWT MPI programs based on D-IS and D-ISSC take advantage of spatial and temporal data localities with optimized asynchronous memory accesses. The split-phase memory access scheme of MPI and D-ISs allows not only long communication latencies overlapping with useful computations, but also lifting the main restriction which conventional (and sequential) computation implies: complete production of data before their consumption. Our D-ISSC provides a software caching mechanism to further reduce the communication latency by caching the split-phase transactions, so that the system would combine the benefits of both latency tolerance and latency reduction together. Although the management of D-IS has a cost, the D-ISSC overcomes this cost and improves the programs performance by eliminating messages. Experimental results show the good speedup of partially evaluated MPI programs and confirm that our concept of distributed image processing optimization is beneficial.

ACKNOWLEDGMENTS

REFERENCES

1. E. Oran Brigham, *Fast Fourier Transform and Its Applications* (Prentice-Hall, 1988).

2. M.-J. Quinn, *Parallel Computing Theory and Practice* (McGraw-Hill, 1994).

3. P.-N. Swarztrauber, "Multiprocessor FFTs," Parallel Computing **5**, 197–210 (1987).

4. R.-M. Chamberlain, "Gray Codes, Fast Fourier Transforms and Hypercubes," Parallel Computing **6**, 225–233 (1988).

5. M. V. Aliev, A. M. Belov, A. V. Yershov, and M. A. Chicheva, "Parallelization of the Hypercomplex Discrete Fourier Transform," Pattern Recognition and Image Analysis, **15** (1), 110 (2005).

6. A.-P.-W. Bohm and R.-E. Hiromoto, "The Data Flow Parallelism of FFT," in *Advanced Topics in Dataflow Computing and Multithreading, Ed. by G.-R. Gao, L. Bic, and J.-L. Gaudiot, ISBN: 0–8186–6542–4* (1995), pp. 393–404.

7. N.-D. Jones, "An Introduction to Partial Evaluation," ACM Computing Surveys **28** (3), (1996).

8. Mogensen and P. Sestoft, "Partial Evaluation," in *Encyclopedia of Computer Science and Technology*, Ed. by A. Kent and J. G. Williams (1997), vol. 37, pp. 247–279.

9. A. P. Ershov, "Mixed Computation: Potential Applications and Problems for Study," Theoretical Computer Science **18** (1982).

10. I. Bjorner, A. Ershov, and N. Jones, *Partial Evaluation and Mixed Computation* (North-Holland, 1988).

11. C. Consel and O. Danvy, "Partial Evaluation of Pattern Matching in String," Information Processing Letter **30** (2), 79–86 (1989).

12. C. Consel and O. Danvy, "Static and Dynamic Semantic Processing," in *ACM Symposium on Principles of Programming Languages* (1991), pp. 14–23.

13. J. Jorgensen, "Generating a Compiler for a Lazy Language by Partial Evaluation," in *ACM Symposium on Principles of Programming Languages, 1992*, pp. 258–268.

14. N. Jones, C. Gomard, and P. Sestoft, *Partial Evaluation and Automatic Program Generation* (Prentice-Hall, 1993).

15. P. Sesyoft and H. Sondergaard, Eds., in *Special Issue on Partial Evaluation and Semantic-Based Program Manipulation (PEPM'94)*, (Lisp and Symbolic Computation, 1995), vol. 8, no. 3.

16. M. Sperber, H. Klaeren, and P. Thiemann, "Distributed Partial Evaluation," in *PASCO'97: Second Int. Symp. on Parallel Symbolic Computation,* Ed. by E. Kaltofen (World Scientific Publishing Company, Maui, Hawaii, 1997), pp. 80–87.

17. J.-L. Lawall, *Faster Fourier Transforms via Automatic Program Specialization* (IRISA Research Reports, 1998).

18. A.-A. Faraj, "Communication Characteristics in the NAS Parallel Benchmarks," *Master Thesis, College of Arts and Sciences* (Florida State University, October, 2002).

19. D. Lahaut and C. Germain, "Static Communications in Parallel Scientic Programs," in *PARLE'94, Parallel Architecture and Languages*, (LNCS 817, 1994), pp. 262–276.

20. R. Gluck, R. Nakashige, and R. Zochling, "Binding-Time Analysis Applied to Mathematical Algorithms," in *17th IFIP Conf. on System Modelling and Optimization*, Eds. J. Dolezal and J. Fidler (Prague, Czech Republic, 1995).

21. Ogawa H., Matsuoka S. "OMPI: Optimizing MPI programs using Partial Evaluation," in *Proc. of the 1996 IEEE/ACM Supercomputing Conf.* (Pittsburgh, 1996).

22. Arvind, R. S. Nikhil, and K. K. Pingali, "I-Structures: Data Structures for Parallel Computing," ACM Trans. on Programming Languages and Systems **11** (4), 598–632 (1989).

23. P. S. Barth, "Using Atomic Data Structures for Parallel Simulation," in *Proc. of the Scalable High Performance Computing Conf.* (VA, Williamsburg, 1992).

24. S. Sur and W. Bohm, "Efficient Declarative Programs: Experience in Implementing NAS Benchmark FT," *Technical Report CS–93–128* (Colorado State University, 1993).

25. X. Shen and B. S. Ang, "Implementing I-Structures at Cache Coherence Level," in *Proc. on the 5th Annual MIT Student Workshop on Scalable Computing* (MIT, 1995).

26. W.-Y. Lin and J.-L. Gaudiot, "I-Structure Software Cache—A Split-Phase Transaction Runtime Cache System," in *Proc. of PACT '96 Boston* (MA, Boston, 1996), pp. 20–23.

27. A. Cristóbal-Salas and A. Tchernykh, "I-Structure Software Cache for Distributed Applications," Dyna, No. 141, 67–74 (1971); Dyna (Medellín, Colombia, 2004), ISSN 0012–7353.2004.

28. A. Cristóbal, A. Tchernykh, J.-L. Gaudiot, and W. Y. Lin, "Non-Strict Execution in Parallel and Distributed Computing," Int. J. of Parallel Programming **31** (2), 77–105 (2003).

29. J.-B. Dennis and G.-R. Gao, "On Memory Models and Cache Management for Shared-Memory Multiprocessors," CSG MEMO 363, CSL, (MIT, 1995).

30. J. N. Amaral, W.-Y. Lin, J.-L. Gaudiot, and G. R. Gao, "Exploiting Locality in Single Assignment Data Structures Updated Through Split-Phase Transactions," Int. J. of Cluster Computing, Special Issue on Internet Scalability: Advances in Parallel, Distributed, and Mobile Systems, **4**, 4 (2001).

31. A. Cristóbal-Salas, A. Tchernykh, and J.-L. Gaudiot, "Incomplete Information Processing for Optimization of Distributed Applications," in *Proc. of the Fourth ACIS Int. Conf. on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing (SNPD'03)* (ACIS, 2003), pp. 277–284.

32. K.-M. Kavi, A.-R. Hurson, P. Patadia, E. Abraham, and P. Shanmugam, "Design of Cache Memories for Multithreaded Dataflow Architecture," in *Proc. of ISCA* (1995), pp. 253–264.

33. R. Govindarajan, S. Nemawarkar, and P. LeNir, "Design and Performance Evaluation of a Multithreaded Architecture," in *Proc. of the 1st Int. Symp. on High-Performance Computer Architecture* (Raliegh, 1995), pp. 298–307.

34. E. Mikulic, "Haar Wavelet Transform," http://dmr.ath.cx/gfx/haar/index.html. 2004.

Andrei Tchernykh received his Ph.D. degree in computer science from the Institute of Precise Mechanics and Computer Technology of the Russian Academy of Sciences (RAS), Russia in 1986. From 1975 to 1995 he was with the Institute of Precise Mechanics and Computer Technology of the RAS, Scientific Computer Center of the RAS, and at Institute for High Performance Computer Systems of the RAS, Moscow, Russia. Since 1995 he has been working at Computer Science Department at the CICESE Research Center, Ensenada, Baja California, Mexico. His main interests include cluster and Grid computing, incomplete information processing, and on-line scheduling.

Alfredo Cristóbal-Salas received his Ph.D. degree in computer science from the Computer Science Department at the CICESE Research Center, Ensenada, Baja California, México. Now he is a researcher at School of Chemistry Sciences and Engineering, University of Baja California, Tijuana, B.C. Mexico His main interests include cluster and Grid computing, incomplete information processing, and on-line scheduling.

Vitaly Kober obtained his MS degree in Applied Mathematics from the Air-Space University of Samara (Russia) in 1984, and his PhD degree in 1992 and Doctoral degree in 2004 in Image Processing from the Institute of Information Transmission Problems, Russian Academy of Sciences. Now he is a titular researcher at the Centro de Investigación Científica y de Educatión Superior de Ensenada (Cicese), México. His research interests include signal and image processing, pattern recognition.

Iosif A. Ovseevich graduated from the Moscow Electrotechnical Institute of Telecommunications. Received candidate's degree in 1953 and doctoral degree in information theory in 1972. At present he is Emeritus Professor at the Institute of Information Transmission Problems of the Russian Academy of Sciences. His research interests include information theory, signal processing, and expert systems. He is a Member of IEEE, Popov Radio Society.