I-STRUCTURE SOFTWARE CACHE FOR DISTRIBUTED APPLICATIONS

ALFREDO CRISTÓBAL-SALAS
School of Chemistry Sciences and Engineering, University of Baja California, Tijuana, B.C. Mexico 22390
cristobal@uabc.mx

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

Recibido para revisar 4 de Octubre de 2003, aceptado 24 de Octubre de 2003, versión final 15 de Noviembre de 2003

RESUMEN: En este artículo, describimos el caché de software I-Structure para entornos de memoria distribuida (D-ISSC), lo cual toma ventaja de la localidad de los datos mientras mantiene la capacidad de tolerancia a la latencia de sistemas de memoria I-Structure. Las facilidades de programación de los programas MPI, le ocultan los problemas de sincronización al programador. Nuestra evaluación experimental usando un conjunto de pruebas de rendimiento indica que clusters de PC con I-Structure y su mecanismo de cache D-ISSC son más robustos. El sistema puede acelerar aplicaciones de comunicación intensiva regulares e irregulares.

PALABRAS CLAVES: I-Structures, Paso de Mensajes, caché de software, transacciones de división de fase

ABSTRACT: In this paper, we describe the I-Structure Software Cache for distributed memory environments (D-ISSC), which takes advantage of data locality while maintaining the capability of latency tolerance of I-Structure memory systems. D-ISSC facilitates the programming of MPI programs hiding synchronization issues from the programmer. Our experimental evaluation using a set of benchmarks indicates that commodity PC clusters with both IS and its caching mechanism D-ISSC are more robust. The system can deliver speedup for both regular and irregular communication-intensive applications.

KEYWORDS: I-Structures, message passing, software cache, split phase transactions

1. INTRODUCTION

Non-Blocking threads and message passing execution models have been proposed as effective means to overlap computation and communication in distributed memory systems. The MPI standard [23] is designed to achieve high performance as well as portability. However, application area of MPI has been somewhat restricted to regular, coarse grained, and computation intensive applications. Some optimization techniques that can be applied to the MPI in order to improve its performance can be found in [8, 18]. These optimization techniques improve the time of sending and receiving messages but they do not reduce the number of messages. All details of programming are still in programmers' hands.

In multiprocessor systems, arbitrary (chaotic) writing and reading may occur because of parallel computations. When processors communicate, they need to synchronize with the purpose of ensuring that valid data are been used and in order to avoid race conditions. If data are

not available, the processor must either wait for the data to arrive, using coarse or fine grain synchronization or switch to another process, and occasionally poll for the data arrival or wait for an interrupt announcing the arrival of the data. I-Structures (ISs) are an attempt at solving this problem [3].

ISs [4, 15, 20, 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. Likewise, any copy of the data elements in a local cache will not be updated. Therefore, the cache coherence is already embedded in the IS memory system. In ISs each element maintains a presence bit that has three possible states: *full, empty* or *deferred*, indicating whether a valid data has been stored in the element. ISs allow the description and use of partially defined information.

Split-phase operations of IS are also used to enable the tolerance of request latencies by a decoupling between the initiators and the receivers of communication/synchronization transactions. 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, fully asynchronous operations, and split-phase memory accesses.

ISs have been implemented in several architectures [1, 5, 12, 22, 23] as a fine-grain synchronization mechanism and as a mean of avoiding the cache coherence problem.

However, data locality is not exploited in the IS memory system and this becomes its major drawback [16]. Research on data locality exploitation using ISs can be found in [15]. 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. This cache system would provide the ability for communication latency reductions while maintaining the communication latency tolerance.

Description of mechanisms of caching single assignment data structures has been presented in [7, 9, 10, 11, 19].

Experimental results, which demonstrate the impact of the I-Structure Software Cache (ISSC) on the non-blocking multithreaded systems, have been presented in [13, 14, 17]. Hit ratios of 90% were archived with a cache block size of 16 for all benchmark programs. ISSC increases the system utilization and improves the overall system performance up to 95%. The *cache in advance* scheme of ISSC also provides the adaptability to the unpredictable communication characteristics in the system. This makes ISSC achieve the same performance without being affected by the variation of the communication latency [2].

We have implemented a Distributed I-Structure Software Cache (D-ISSC) and evaluated its performance in a NUMA S2MP ORIGIN2000 and in a cluster of PCs. In this paper, we discuss the design of this cache system.

In the next section, details of our D-ISSC implementation are described. In section 3, we present some experimental results. Lastly, some conclusions are discussed in section 4.

2. D-ISSC

In previous section we show that the I-Structure memory system has been adopted in non-blocking multithreaded systems for its ability to hide the latency of accessing I-Structure elements by the split-phase transaction mechanisms. However, a major drawback of split-phased transactions is that data locality of* remote data is not utilized by conventional on-board cache systems. On the other hand, message passing execution model in high-performance computing is gaining acceptance with the raising of clusters. In this model, inter-process communications are performed using split-phase transactions. Thus, the I-Structure memory system can be easily implemented under this model. Nevertheless, the need of explicit synchronization points between read and writes operations impose unnecessary burden on programmer's hands. In addition, the manual usage of temporal and spatial data locality has restricted the applicability of message passing paradigm to coarse-grained computations.

D-ISSC is further development of the ISSC memory system. It provides a software caching mechanism for ISs in distributed memory systems. The D-ISSC works as an interface between the MPI processes and the network interface. It intercepts all the read operations to the IS memory system. A remote memory request is sent out to the remote process only if the requested data are not available in the local software cache.

Let us consider some features of the D-ISSC.

Unique ID to identify IS inside the local memory system. Even though the single assignment rule of IS simplifies the cache coherence problem, some cache coherence problems would still occur when the IS memory space is de-allocated and re-utilized. To totally avoid the cache coherence problem, a logical address, like the data structure ID, has to be used. It is the job of the compiler to make sure that no two data structures have identical Ids.

Remote IS accesses use the MPI standard library. MPI makes our cache system portable and it also provides non-blocking communication facilities. In non-blocking communication, the processes call a MPI routine to set up a communication (send or receive), but the routine returns before the communication has been completed. Non-blocking call can be used to avoid deadlock situations. Also, they are used to overlap communication and computation. These facilities allow the latency hiding. Interprocess communications in MPI are made by split-phase transactions. Thus, accesses to IS elements can be naturally implemented. D-ISSC gives additional properties to the split-phase feature of MPI standard. Our system can completely execute a send or receive operations irrespective of availability of data.

Cache block is allocated in advance. In conventional cache designs, the cache space is allocated when the data block is brought back to the local process. However, in the D-ISSC, the cache space is allocated in advance when a read miss is detected. Indeed, due to the long latency and unpredictable characteristics of the network in distributed memory systems, a second remote access to the data elements in the same data block may be issued while the first request is still traveling through the network.

Centralized queues management. A simple centralized method is used to implement the queues of deferred requests. The length of the queue for each IS element is bounded by the number of processes in the system. This is because only one request is sent from each process. Future deferred reads are kept locally in the process.

Associative cache memory. Cache search schemes play a very important role in the cache performance. In hardware design, a fully associative cache has the highest performance because of its parallel search and the full utilization of the cache space. A cache line could be directly accessed by index addressing. The tag of a cache block is stored in the cache entry along with the index of the cache line. A cache entry is searched in the hash table using the tag field.

The index field of a cache entry stores the index of the cache line allocated to this data block. As in a fully associative cache, this means that a data block could be mapped to any cache line.

Temporal and spatial exploitation. When a remote memory request occurs, the D-ISSC not only requests just one element but also requests the entire block of data surrounding the element. Therefore, spatial data locality can also be exploited. Temporal data exploitation refers to the reuse of data that are already in cache.

Cache coherence capability. The single assignment property of the IS memory system enables the ISSC as a software cache without any hardware support. Because of the inherent cache coherence feature of ISs, no cache coherence problem is encountered in the IS cache design. This significantly reduces the overhead of the cache system.

Write-through policy. A simple write-through policy is adopted to guarantee the services of the deferred requests in the ISs. It also ensures the legality of write operations for a single assignment memory system. In other words, there is no caching for write operations.

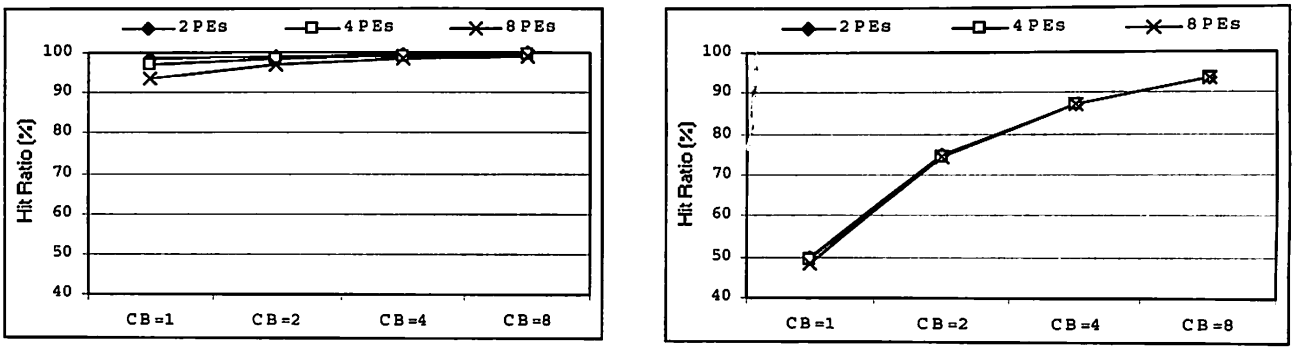
3. EXPERIMENTAL RESULTS

Experimental results are presented for SGI ORIGIN2000 with 10 MIPS R10000 processors running at 195MHz with 1280MB of main memory and a network bandwidth of 800MBs/sec with a hypercube topology; and 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 well-known dense matrix multiplication (MM) and conjugate gradient (CG) algorithms.

Table 1. Number of messages of IS and D-ISSC programs for the MM and CG benchmarks varying the number of processors.

Program	Implementation	Number of messages		
		2 PEs	4 PEs	8 PEs
MM	IS	2097152	3145728	3670016
	D-ISSC (CB=1)	32768	98304	229376
CG	IS	134144	205824	250880
	D-ISSC (CB=1)	67584	104448	129024



(a) (b)

Figure 1. Hit ratio of the MM (a) and CG (b) programs for different number of processes and different cache block sizes.

In the MM, each 128x128 matrix is represented by a IS with double precision values.

The MM has excellent temporal locality of data reference: two input matrices are constantly referenced during the computation. Block distribution is used in this benchmark i.e. matrices are distributed by rows and columns. Data locality is reduced when processing elements are added due to initial data distribution.

The steepest descent method, also known as the gradient method, is a simplest example of gradient-based methods.

This loose coupling algorithm solves a system of 256 equations with 256 variables. Matrices are also represented by ISs with double precision values, and an interleave data distribution is used.

The performance of benchmarks with two different MPI implementations has been compared:

(1) *IS.* Program uses IS memory system without cache support

(2) *D-ISSC.* The D-ISSC system is used.

3.1 MESSAGE REDUCTION

Table 1 shows the number of messages of the IS and D-ISSC programs for MM and CG benchmarks obtained for different number of processes. We see that for both programs the total number of messages is increased when more processes are added. It is a result of data

distribution between processes. We can also see an impact of our D-ISSC in reducing the number of messages in the system. The number of messages of MM is reduced significantly by a factor of more than 16 compared to the original quantity of *IS* program without D-ISSC support, while the number of messages of CG is decreased only by a factor of 1.9.
This difference is because the cache allows processing more than 90% of all requests locally in MM and only 48% in CG. Hit ratio of the MM and CG programs for different number of processes and different cache block sizes is shown in Figure 1.

3.2 SPEEDUP

Figure 2 shows the speedup obtained by parallelization of benchmark programs for different number of processes (1, 2, 4, and 8). In this figure, we present speedups obtained when comparing IS and D-ISSC programs, with different cache block sizes (1, 2, 4, and 8 IS elements), against the IS program running in just one process.
We see that IS programs cannot achieve speedup more than one in all cases. These programs do not exploit data locality (temporal and spatial). Speedup gained by parallelization is rather low. IS programs perform several memory accesses, local and remote, without any internal data buffering or requests made by blocks.
In Figures 2a and 2b we see that the speedup of the D-ISSC is significant for MM because D-ISSC efficiently exploits temporal and spatial data locality of the algorithm. For instance, with eight PEs speedup goes from 4.1 with CB=1 to 5.4

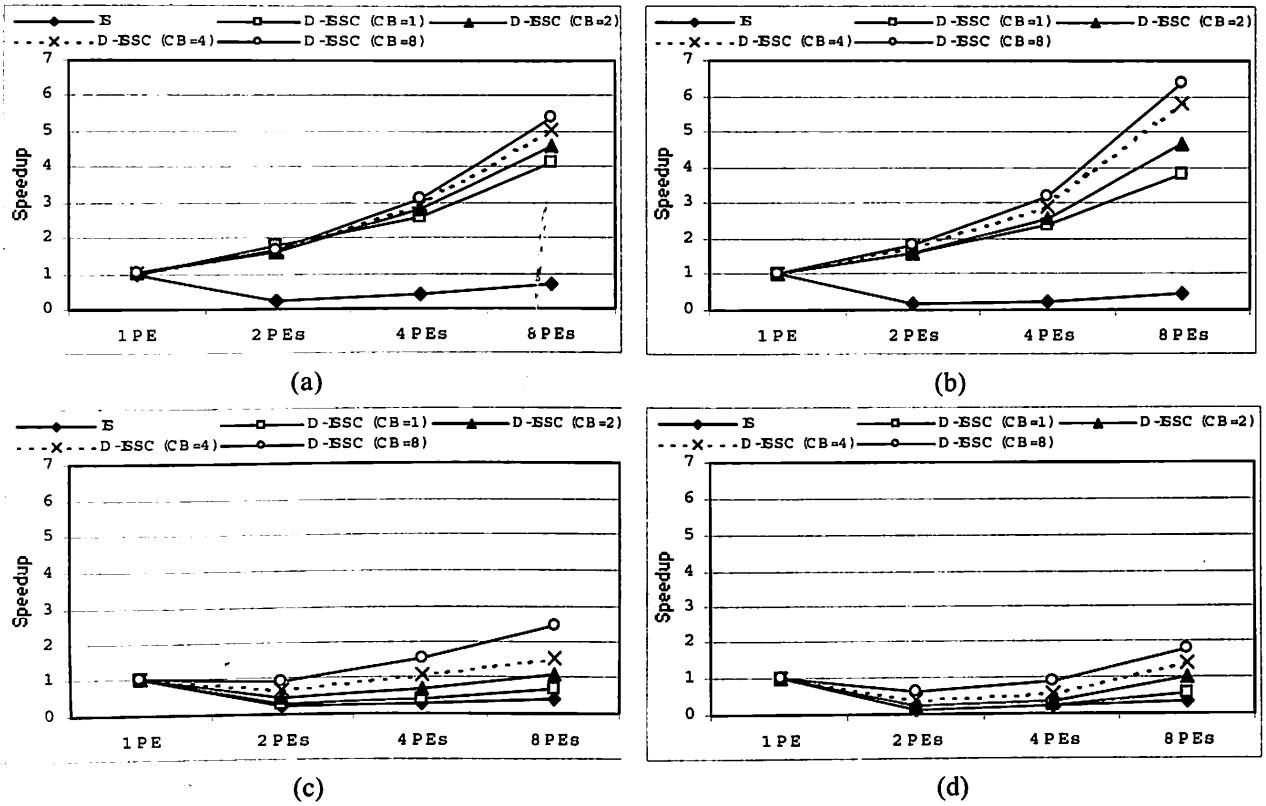


Figure 2. Speedup measurements for MM benchmark program in ORIGIN (a) and Cluster (b) and for CG in ORIGIN (c) and Cluster (d) with different number of PEs.

When CB=8 in Origin 2000. For cluster, speedup varies from 3.8 to 6.4 with cache blocks 1 and 8 respectively. CG (Figures 2c and 2d) has low re-using of data that impact in its lower

speedup. For instance, for eight PEs and increasing cache block sizes from one to eight, speedup varies from 0.7 to 2.5 in Origin 2000 and from 0.6 to 1.8 in our cluster of PCs.
Figure 3 shows the speedup obtained by parallelization of programs with D-ISSC on different number of processors for MM and CG. For number of processors more than one, significant reduction of the number of messages can be obtained in MM with D-ISSC (see Table 1). D-ISSC, with eight PEs, makes it at least six times faster in Origin 2000 than corresponding IS program. This happens when cache block size is equal to one (Figure 3a).
For our cluster, using the same configuration, PE=8 and CB=1, MM is more than eight times faster (Figure 3b).
Less data locality exploitation in CG results in a lower speedup of the program with D-ISSC. CG with D-ISSC is at least 1.02 times faster than without D-ISSC in both computers (Figures 3c, 3d). Results are obtained for PE=2 and CB=1.

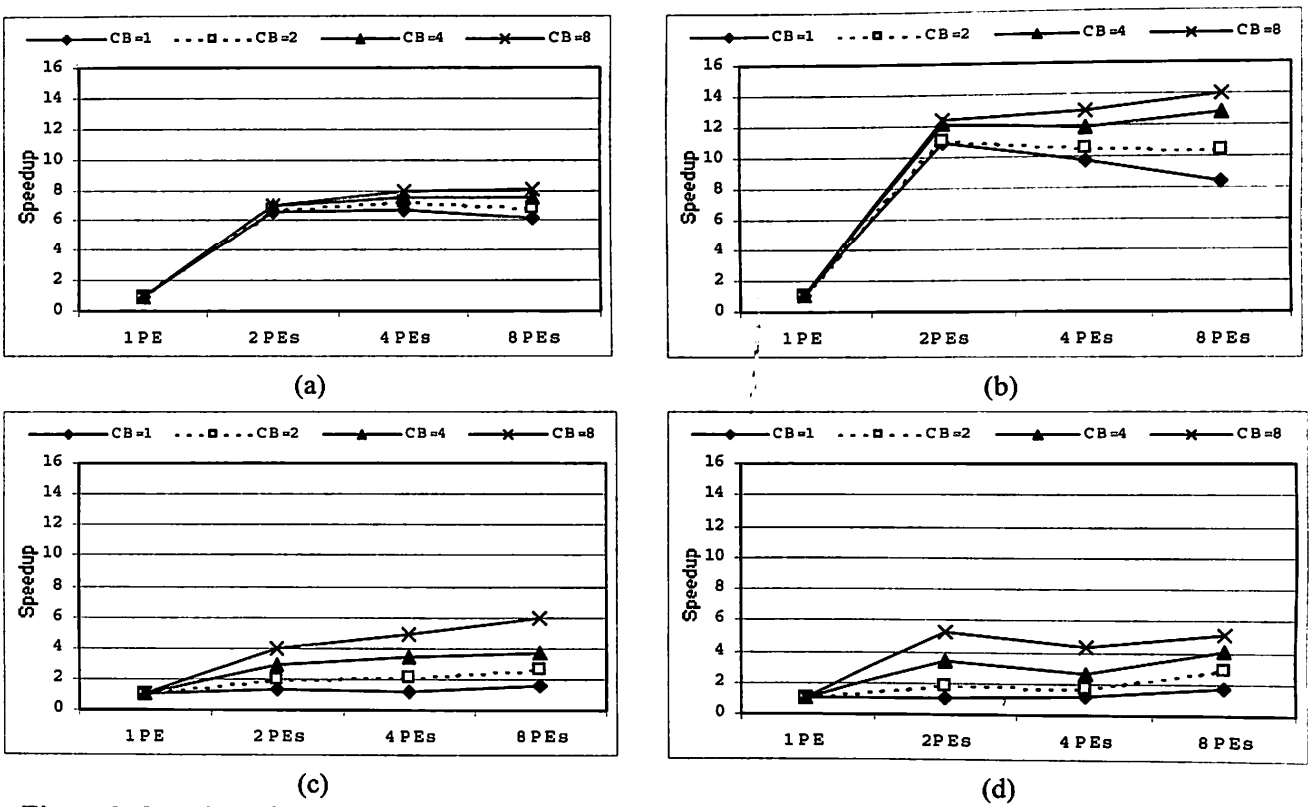


Figure 3. Speedup of D-ISSC over IS, for MM (a,b) and CG (b,d) benchmark programs varying the number of processors and the cache block size. Results are presented for Origin2000 (a and c) and for a cluster of PCs (b and d).

4. CONCLUSIONS

In this paper, the design, implementation and experimental results of the D-ISSC for PC clusters and ORIGIN 2000 have been presented. We have shown that in a distributed memory systems, the split-phase memory access scheme of MPI and IS allows an overlap between long communication latencies and useful computations. In addition, D-ISSC provides a software caching mechanism to further reduce the communication latency by caching the split-phase transactions, so the system would combine the benefits of latency tolerance and latency

reduction. It is more significant for systems with large latency such as PC clusters with commodity NICs. By exploiting both the temporal and the spatial global data locality, the number of inter-process communication decreases dramatically. D-ISSC not only helps the system by exploiting the data locality in different type of applications, but it also reduces the traffic in the network. The distributed D-ISSC memory system significantly reduces the network latency and makes the system more scalable.

We have shown that even though D-ISSC incurs an additional overhead for its operations, by taking advantage of the global data locality in applications and with the amount of communication interface overhead saved by D-ISSC, the D-ISSC improves system performance in both parallel systems

An additional optimization technique based on partial evaluation applied to D-ISSC in order to further reduce number of messages is presented in [6]. Our main results can be summarized as follows:

1. D-ISSC support in the MPI results in increased robustness to latency variation. The speedup obtained with D-ISSC increases for bigger number of processor elements.
2. The D-ISSC significantly reduces the amount of traffic in the network.
3. The performance of the programs with D-ISSC support is improved for all benchmarks for PC cluster and ORIGIN2000.

5. REFERENCES

1. B. S. Ang, Arvind, and D. Chiou. "StarT the next generation: Integrating global caches and dataflow architecture". In G. R. Gao, L. Bic, and J.-L. Gaudiot, editors, Advanced Topics in Dataflow Computing and Multithreading, pages 19--54 . IEEE Comp. Soc. Press, 1995.
2. J.N.Amaral, W-Y.Lin, J-L. Gaudiot, and G.R.Gao. Exploiting Locality in Single Assignment Data Structures Updated Through
3. 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 28-30, 1991
4. Cristobal A., Tchernykh A., Gaudiot J-L., Lin W-Y "non-strict execution in parallel and distributed computing". Submitted to International Journal of Parallel Programming, February 2002.
5. J. B.Dennis and G. R.Gao. "On memory models and cache management for shared-memory multiprocessors", CSG MEMO 363, Laboratory for computer science, MIT., March 1995.
6. T. von Eicken, D.E.Culler, S.C.Goldstein, and K.E.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.
7. J-L Gaudiot and C-T Cheng. "A scalable cache design for I-Structures". In proceedings of the International Conference on Parallel Processing, August 1996.
8. R.Govindarajan, S.Nemawarkar, P.LeNir. "Design and performance evaluation of a multithreaded architecture", In proceedings of the first international symposium on High-Performance Computer Architecture, Raliegh, pp. 298-307, 1995.
9. K.M.Kavi, A.R.Hurson, P.Patadia, E.Abraham, and P.Shanmugam. "Design of cache memories for multithreaded dataflow architecture". In ISCA 95, pages 253-264, 1995.
10. David Kranz, Beng-Hong Lim, Anant Agarwal, and Donald Yeung. "Low-cost support for fine grain synchronization in multiprocessors". Kluwer Academic publishers, 1994.
11. W-Y.Lin, J.N.Amaral, J-L.Gaudiot, and G.R.Gao. Caching Single-Assignment Structures to Build a Robust Fine-Grain Multi-Threading System. Technical report,

Dept. of E.E. - Systems, University of Southern California, July 1999.
12. W-Y. Lin, J.N.Amaral, J-L.Gaudiot, and G.R.Gao. Caching Single-Assignment Structures to Build a Robust Fine-Grain Multi-Threading System. International Parallel and Distributed Processing Symposium, Cancun, Mexico May 1-5, 2000
13. W-Y.Lin, J-L.Gaudiot. "I-Structure Software Cache – A split-Phase Transaction runtime cache system", Proceedings of PACT '96 Boston, MA, Oct. 20-23, 1996.
14. W-Y.Lin, J-L.Gaudiot. "The design if an I-Structure Software Cache System", Proceedings of MTEAC'98, Las Vegas, Feb 1-4.
15. W-Y.Lin, J-L.Gaudiot, J.N.Amaral, and G.R.Gao. Performance Analysis of the I-Structure Software Cache on Multi-Threading Systems, 19th IEEE Internation Performance, Computing and Communication Conference, IPCCC2000, Phoenis, Arizona, Feb. 20-22, 2000
16. H. Ogawa, S. Matsuoka, OMPI: Optimizing MPI programs using Partial Evaluation, Proceedings of the 1996 IEEE/ACM Supercomputing Conference, Pittsburgh, November 1996.
17. G.M. Papadopoulos. Implementation of a General-purpose Dataflow multiprocessor. PhD thesis, Laboratory for Computer Science, MIT, August 1988.
18. X.Shen, B.S.Ang "Implementing I-Structures at cache coherence level" In proceedings on the 5th annual MIT student workshop on scalable computing, MIT, 1995
19. S.Sur, W.Bohm "Efficient declarative programs: experience in implementing NAS Benchmark FT", Technical Report CS-93-128, Oct 1993. Colorado State University.
20. Kevin B. Theobald, Olivier Maquelin, Guang R. Gao, et al. Overview of the Threaded-C Language, Technical Memo 19, CAPSL, University of Delaware, March 16, 1998.
21. Tatebe Osamu, Kodama Yuetsu, Sekiguchi Santoshi, Yamaguchi Yoshinori "Highly efficient implementation of MPI point-to-point communication using remote memory operations" Proceedings of 12th ACM ICS98, pp.267-273, July 1998, Melbourne, Australia.