ICOT Technical Report: TR-769

TR-769

# Applying Inter-Cluster Shared Memory Architecture to a Parallel Inference Machine

by
H. Sakai, A. Nakase, T. Takewaki
& S. Asano (Toshiba)

April, 1992

# APPLYING INTER-CLUSTER SHARED MEMORY ARCHITECTURE TO A PARALLEL INFERENCE MACHINE

Hiroshi Sakai, Akihiko Nakase, Toshiaki Takewaki, Shigehiro Asano

Information Systems Laboratory, Toshiba Research and Development Center
1 Komukai Toshiba-cho, Saiwai-ku Kawasaki-shi, Kanagawa 210, Japan

This paper discusses the feasibility of a parallel inference machine with the inter-cluster shared memory architecture in stead of the message passing mechanism. The discussion includes two possible memory architectures which fit to the memory access features, and the system software under the proposed memory acrchitecture. Although the main objective is the smaller latency of the remote memory access, additional advantages are also discussed.

## 1 Introduction

Parallel processing is one of the most important keys for faster execution in the knowledge engineering domain and there are a lot of research activities on parallel processing models, parallel programming languages, and machines with parallel architecture.

Under Japan's Fifth Generation Computer Project, Parallel Inference Machines (PIMs) are being developed [1]. They employ a common parallel logic programming language KL1 which was derived from Flat GHC [5].

The machine architectures have about 1,000 processor elements (PEs). Most of the machines consist of clusters, each of which has several PEs and a shared memory. The PEs of a cluster can read from and write into the shared memory by load and store instructions. However, between clusters no memory is shared. Therefore, when a PE requires a new goal from a different cluster and/or refers to an object stored in a different cluster, the PE has to send a request message. We will call it the inter-cluster *message passing* architecture.

This paper discusses the feasibility of another inter-cluster connection where a PE can refer to a different cluster's memory (a remote memory) by load and store instructions. We call it the inter-cluster *shared memory* architecture. The rest of this paper consists of the following discussions. First the overview of KL1, its parallel execution model, and its memory access features are described. Secondly the current PIM architecture and its KL1 system are reviewed. Then two possible ways of the proposed memory architectures, that is the hierarchical cache memory and the memory access based inter-cluster network, are discussed. Finally the KL1 system adjusted to the proposed memory architecture is considered.

This research was conducted under the Japan's Fifth Generation Computer Project and the results will be applied to the PIM/k being developed by the authors.

## 2 KL1 and its Parallel Execution

### 2.1 Overview of KL1

A KL1 program consists of clauses as shown in figure 1. A clause consists of a head, an "is-implied-by" operator, a guard, a commit operator, and a body. The guard and

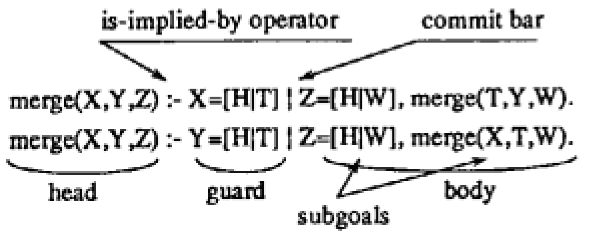
```
merge(X,Y,Z) :- X=[H|T] | Z=[H|W], merge(T,Y,W).
merge(X,Y,Z) :- Y=[H|T] | Z=[H|W], merge(X,T,W).
```

Figure 1 KL1 program

the body may contain several subgoals separated by commas which mean AND operators. Subgoals are built-in (e.g. arithmetic operators) or defined in the program. A guard must consist of built-in subgoals only, while the body may contain subgoals of the both types.

During the program execution, goals are evaluated by head unification with the various clause heads in the system. If unification can take place without the instantiation of goal variables and the clause guard succeeds, the clause is selected. There are three possible outcomes.

- If an appropriate clause is selected, the body of the clause is processed.
- If no appropriate clauses are found but some clauses chance to become appropriate by instantiating variables in them, then the evaluation of the goal is suspended until relevant instantiation takes place.
- Otherwise the whole program results in failure, because all the goals are connected by the AND operators.

As for the execution of the body, all its subgoals can be evaluated in parallel. A subgoal defined in the program will create a new goal. A built-in subgoal will be evaluated immediately if the operands are instantiated enough, otherwise its evaluation will be suspended.

### 2.2 Parallel Execution Model

The granule of the parallel processing is a goal. Given enough PEs, all the goals can be evaluated simultaneously.

A goal is represented as a goal record which includes appropriate arguments. If an argument is a number or an atom which occupies one word, it is stored in the goal record. If it is a variable, a pointer to the variable is stored because a variable may be shared among goal records and be changed by their evaluation. If an argument is a vector which contains several arguments, a pointer to it is stored for the sake of efficiency.

A PIM runs several programs simultaneously and some part of a program can be evaluated with higher priority than the rest of the program, hence in the machine, several goal pools are maintained. The machine has another set of pools for suspended goals. However, for the sake of simplicity the parallel execution model described here has only one goal pool and one suspension pool. During the program evaluation, each PE carries out the procedure shown in figure 2.

Since the procedure requires no message passing mechanism explicitly and a goal reduction might refers any KL1 object, the best memory architecture for a KL1 system would be a large memory shared by a number of PEs with short access time. However It seems impossible from the hardware point of view, we have to settle for a possible one.

## 2.3 Memory Access Features

For the memory access features, KL1 is different from Prolog, the sequential logic programming language. Prolog employs back tracking in its execution model. The control structures and most of the variables can be allocated in a stack, hence a heap space, the memory pool kept for dynamic memory allocation is used only to store the data which may outlive the control structure which creates them. As a rule, using a stack is more efficient than using a heap space because the higher memory access locality near the stack top increases the cache hit ratio and decreases the garbage collection (GC) frequency.

On the other hand, KL1 program execution causes heavy bus traffic, which can be estimated under the following assumption.

- The performance of a single PE is N reductions per second.
- The number of PEs in a cluster is P.
- The system performance increases in proportion to P.
- One reduction consumes the heap space by W words.
- The cache block size is C words.
- A cache miss takes place only when an object is stored into the heap space, which takes two bus transactions. That is, the contents of the relevant cache line must be stored into the shared memory before the contents of the heap space is loaded.
- Each PE has its own heap space so that a PE doesn't interfere other PEs' cache memories.

This too optimistic evaluation concludes that the memory bus traffic is $\frac{2 \times N \times P \times W}{C}$ transactions per second. Under the set of values ($N = 500{,}000$, $P = 8$, $W = 2$, $C = 4$) which seems typical to the current PIMs, the memory bus traffic will be 4,000,000 transactions per second. Therefore, from the memory architectural point of view, it seems the most important to resolve the memory bus bottle-neck caused by the heap consumption.

```
A PE tries to get a goal record from the goal pool.
If the PE gets a goal record,
then /* goal reduction starts */
  The PE checks the head and the guard of alternative
  clauses one at a time to find out an appropriate one.
  If an appropriate clause is found
  then /* body part evaluation starts. */
    For each subgoal of the body part do
      If the subgoal is defined by the program
      then
        The PE creats a new goal record with appropriate
        arguments and put it into the goal pool.
      else /* a built-in subgoal */
        if the subgoal can be evaluated immediately
        then
          The PE evaluates it.able.
          If the evaluation causes any suspended goals
             to turn evaluable
          then The PE puts them into the goal pool.
          fi
        else
          if the subgoal is not evaluable because some of
            its variables have not been instantiated yet
          then
            the PE creates a new goal record called D-code
            and puts it into the suspend pool.
          else /*the evaluation fails */
            The PE reports the whole program has failed.
          fi
        fi
      fi
    endfor
  else
    If there is a chance that one of the clauses may turn
       appropriate by instantiating some variables
    then The PE put the goal record into the suspend pool.
    else /*the evaluation fails */
         The PE reports the whole program has failed.
    fi
  fi
else
  if there are some busy processors in the system.
  then The PE waits shortly and go to the first statement.
  else
    If the suspend pool holds some goal records
    then The PE reports a dead lock has taken place.
    else The PE reports the whole program has terminated
           successfully.
    fi
  fi
fi
```

Figure 2 Goal Reduction Procedure of the KL1 system

## 3 Typical Parallel Inference Machines

The current PIMs with the inter-cluster message passing architecture are based on the following design considerations.

- Reducing the bus traffic within a cluster by optimizing the coherent cache protocol and employing high bandwidth memory bus.
- Enhancing the inter-cluster data transfer bandwidth by increasing the number of channels among clusters
- Curbing the inter-cluster communication frequency by designing the KL1 system software to draw distinctions between inner and inter-cluster operations

### 3.1 Memory Architecture within a Cluster

For a coherent cache protocol, [2] proposed and evaluated the PIM cache. They adopted a copyback protocol because KL1 benchmarks indicate that data write frequency is 36%, which is higher than in conventional languages. To reduce the heavy bus traffic caused by the cache swap in/out, the PIM cache introduced the direct write command which avoids a fetch-on-write strategy. This command can be used to create new data structures on the top of the heap area and the bus commands would reduce half in number on that operation.

### 3.2 Inter-cluster Message Passing Architecture

In spite of employing the PIM cache, the bus traffic would be the bottle-neck in cases that more than ten PEs share a memory through a single bus, yet the final goal is to realize a PIM with 1,000 PEs. Typical PIMs solve this problem by employing inter-cluster network. For example, PIM/p has 8 PEs in each cluster and 64 clusters are connected by a hypercube network. The KL1 system requires that the network should support one to one and one to all communications between clusters.

### 3.3 KL1 System on a Typical PIM

#### 3.3.1 External Data Reference

The KL1 system handles internal pointers and external ones as different data types. They are given different tags, REF and EXREF. A REF pointer holds only a memory address. The KL1 object pointed by a REF pointer can be simply referred by a load instruction.

However, the external reference mechanism is rather complicated as illustrated in figure 3. It is called to export data for a cluster to give another cluster a reference to its KL1 object and it is called to import data for a cluster to get the information. The export table holds the memory address of all exported data, and the import table holds the pairs of the cluster number and the export table entry number of imported data. An EXREF pointer holds the entry number of the import table. A cluster can move a KL1 object without suspending other clusters' goal reduction in favor of these tables. Now we will explain a basic operation on an external reference.

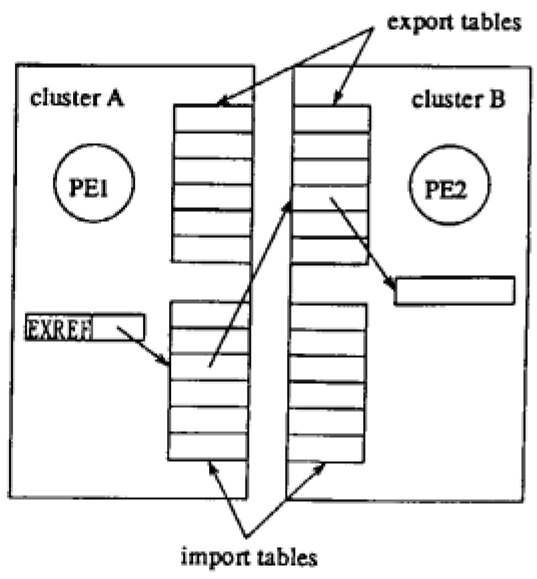

Figure 3 External Data Reference Mechanism of the Current KL1 system

When a PE encounters an EXREF pointer in a dereference operation, it tries to get the KL1 object referred by the pointer. This read operation consists of the following steps. Here we assume that PE1 of cluster A tries to read the KL1 object within cluster B.

- PE1 looks up the proper entry in its import table and sends a read request to cluster B.
- A PE2 of cluster B receives the read request, looks up the proper entry in its export table, gets the memory address of the required object, and sends back a reply message which holds the object.
- PE1 receives the reply message and resumes the dereference operation.

If PE2 finds the KL1 object is a vector which contains pointers, a more complicated procedure takes place. Here we will explain the case that a vector contains a REF pointer. PE2 searches its export table for the entry which holds the same address as that pointer. If it is found, PE2 sends the entry number as part of the reply message as well as the vector itself. Otherwise the cluster has to get a new entry and set the object address in it before sending back the reply message. Then PE1, receiving the reply message, searches its import table for the entry which holds the same cluster-id and the same entry number. If it results in success, the cluster A replaces that part of the vector with the proper EXREF pointer. Otherwise the cluster A has to get a new entry and set the cluster-id and the entry address.

The actual latency of an external reference was reported in case of the Multi-PSI system [6]. In this evaluation, a passive unification across clusters which includes one read message and reply message exchange, takes 23 append logical instruction time, while the same operation within a cluster costs less than one append logical instruction time.

### 3.3.2 Garbage Collection

The KL1 system employs both a real-time GC and a stop-and-copy GC. For the real-time GC, the KL1 system employs a multiple reference bit (MRB) which identifies single and multiple references [4]. The mechanism reflects the observation that most of the objects are single referenced. An object with a single reference can be reclaimed when it turns unreferenced by reclaiming the pointer to it. It is said that if no real-time GC is employed, a typical goal reduction consumes five words of the heap space and the MRB-based GC reduces it to two words. The MRB mechanism also ensures the constant time updating operation of a vector with a single reference. Without any mechanisms which identifies a single reference, the vector would have to be copied at the expense of time proportional to its size because KL1 avoids destructive operations.

Since the real-time GC does not reclaim all the reusable area, a stop-and-copy GC is invoked when the entire heap space of a cluster has been exhausted. The external reference mechanism of the import and the export tables ensures that a cluster can do a stop-and-copy GC without suspending other clusters. However, if a cluster fails to reclaim any area, then all the clusters participate in a global GC which may reclaim some more area by reclaiming external references.

## 4 Possible Inter-cluster Shared Memory Architectures

This section discusses two possible inter-cluster shared memory architectures for a parallel inference machine. One is a hierarchical coherent cache memory [10] as illustrated in figure 4, and the other is a memory access based inter-cluster network as shown in figure 5.

### 4.1 Hierarchical coherent cache memory

In this approach, the first cache memories of a cluster share a second cache memory through a cache bus and the second cache memories share a memory through a memory bus. Although figure 4 illustrates a two-level cache memory, it is possible to extend it to $n$-level.

The most important property of it is the multi-level inclusion which avoids unnecessary bus transactions. This property requires that a second cache memory holds all the cache-line data each of which is contained in one of its relevant first cache memories. In figure 4, if PE11 and PE22 have the data of memory address A, then the second cache memory1 and 2 must have the entry. Now we assume that PE11 tries to store a value at memory address A.

The first cache memory11 (the one owned by PE11) refers its tag memory, finds that there may be valid copies in

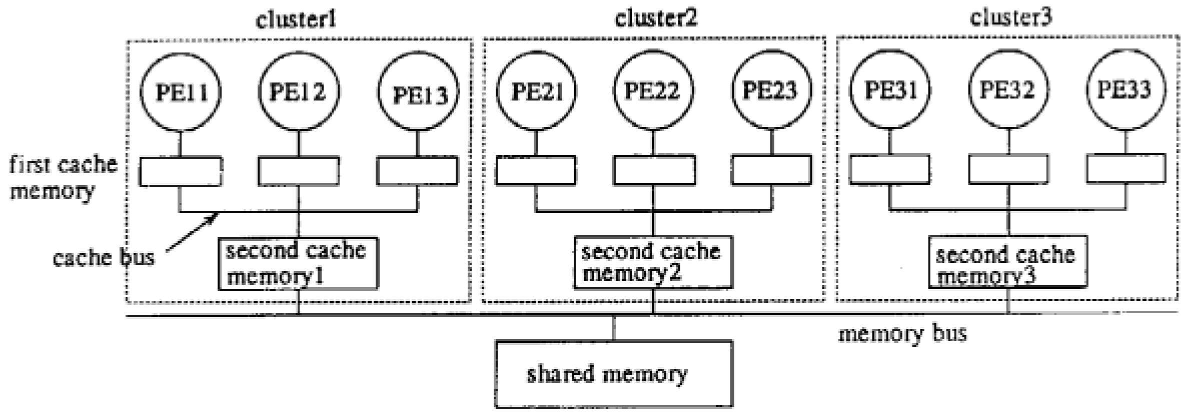

Figure 4 Heirarchical Coherent Cache Memory

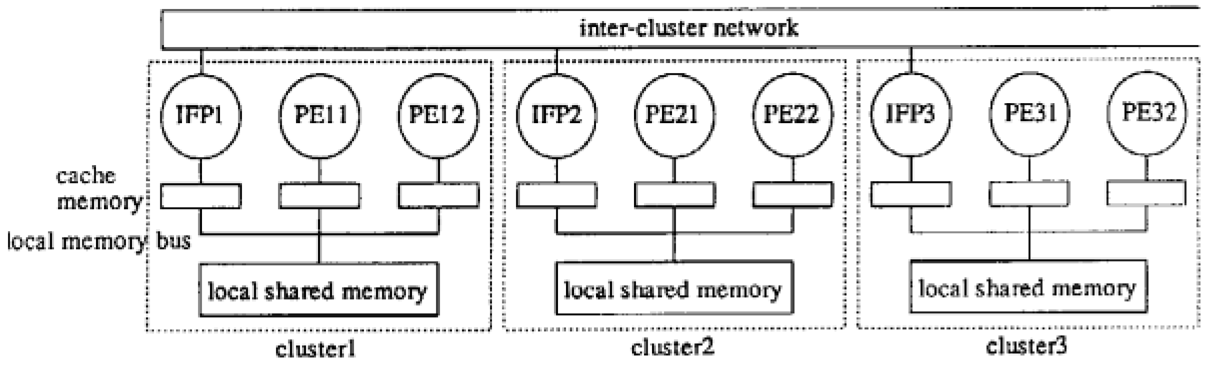

Figure 5 Memory Access Based Inter-cluster Network

other cache memories, and then broadcasts an invalidation command through the cache bus1. The second cache memory1 receives the invalidation command, refers its tag memory, finds there may be valid copies in other second cache memories, and then broadcasts an invalidation command through the memory bus. The second cache memory3 finds no entries and sends back acknowledgements immediately, while the second cache memory2 finds an entry and broadcasts an invalidation command through the cache bus2. In this example, the multi-level inclusion property prevents the cache bus3 from being used by the store operation.

As is discussed in the previous section, it is extremely important to reduce the bus traffic caused by the heap consumption. It would be realized if a second cache memory is large enough so that the heap space of a cluster could be stored in it. Then a store operation to the heap space would cause no memory bus commands because the second cache memory knows that there are no valid entries outside the cluster. The memory bus would be used only when an external memory reference takes place.

### 4.2 Memory Access Based Inter-cluster Network

In this approach, PEs of a cluster share a local shared memory through a local memory bus. Each local shared memory must have unique address across the clusters and an inter-cluster network transfers remote memory access requests between clusters. In figure 5, an interface port (IFP) with a coherent cache memory is attached to each cluster and is connected to the inter-cluster network.

When the PE11 tries to refer to its local shared memory, the cache memories works as usual.

When the PE11 tries to refer to a remote memory, the cache memory11 passes the request to the IFP1's cache memory through the memory bus. Then the cache memory passes the request to the IFP1, which in turn puts the request into the network. If the request is a read, the resultant data will be sent back to the IFP1 and then will be passed to the client PE11 without keeping the data in the cache memories. When the IFP1 receives a memory access request from another cluster, it issues a read or write request to its cache memory as if it were its own request.

Under this architecture, clusters are supposed to use their local shared memories as its heap space. This mechanism resembles the memory architecture of DASH multiprocessor [7] except that our mechanism doesn't employ a directory memory. The difference comes from our estimation that a PIM would have little data access locality across clusters.

The former approach seems to have two advantages. One is that a remote memory access would take less time and the other is that it seems more appropriate as a general-purpose shared memory architecture. The latter approach also seems to have two advantages. One is that the amount of required memory would be less and the other is that the inter-cluster network can be designed to fit the memory access traffic between clusters.

## 5 KL1 System under the Proposed Memory Architecture

### 5.1 Direct Access to the Remote Objects

To exploit the proposed memory architecture, it is important to take the advantage of the low latency of the remote memory access. From this point of view, the import and export tables of the current KL1 system should be avoided because the maintenance takes longer than the remote memory access itself.

Avoiding these tables has two more advantages. One is that the KL1 system would be reduced half in code size, which, in turn, would lead to higher hit ratio of the instruction cache memory. The part which could be removed includes the operations related to external data types, most part of the goal distribution mechanism across clusters as well as the import and export table maintenance. The other advantage is much easier debugging of the system. Debugging of a parallel inference machine with message passing mechanism is difficult because it is necessary to care the states of all the clusters. Since direct access mechanism implies the consistent memory image across clusters, we have only to watch one cluster.

### 5.2 Garbage Collection

Now we have to consider if the avoidance of the import and export tables causes any drawbacks. Avoiding external data types seems not to lead to any difficulties. An operation to these types is executed more efficiently and results in the state equivalent to the case of the current KL1 system.

The real-time GC seems not affected in an adverse way. The MRB mechanism works well to reclaim unreferenced area whether it is inside or outside the cluster. The area is put into the free list of the cluster where it exists.

The stop-and-copy GC, however, is affected significantly. A cluster would not be able to do it independently, because it may move objects which are still referred from other clusters without adjusting the external pointers. Therefore, a cluster which begins to do it must have all the clusters join it cooperatively, which is likely to cause an efficiency problem. The required time for a stop-and-copy GC may well be thought proportional to the amount of live objects and independent of the span of the GCs. Therefore its frequency loses the overall efficiency of the system.

There seems three possible solutions to reduce its frequency and we think their combination would solve the problem. One solution is to substitute the MRB-based real-time GC for a more powerful one like an ordinary reference count mechanism. It would reclaim almost all garbage and reduce the GC frequency. Another solution is just to suspend the goal reduction of the clusters which have exhausted their heap spaces. Since other clusters can continue the goal reduction, the overall efficiency would be acceptable if the number of suspended clusters remains small. A counter idea is to allow a cluster which has exhausted its heap space to use another cluster's heap space. But it doesn't seem any good because every goal reduction would cause several remote memory accesses. The other solution is to balance the amounts of available heap spaces across clusters by moving objects between them during a global GC. It would balance the time when clusters come to exhaust their heap spaces. Related to this solution, it seems appropriate to migrate an object which is referred by only one external pointer into the cluster where the pointer exists.

## 6 Conclusion

This paper discusses the feasibility of a parallel inference machine with the inter-cluster shared memory architecture instead of the message passing mechanism. For the memory architecture, a hierarchical coherent cache memory and the memory access based inter-cluster network seems possible. Although the major advantage is the smaller latency of the remote memory access, it would also reduce the KL1 system code size, which in turn would lead to the higher instruction cache hit ratio and easier debugging. Although the stop-and-copy garbage collection is likely to cause an efficiency problem, the combination of the three possible solutions would solve the problem.

Acknowledgement

We are grateful to Dr. Taki and Dr. Hirata of ICOT who have been leading the PIM project.

## References

[1] Goto,A., et.al., Overview of the Parallel Inference Machine Architecture (PIM), Proceedings of International Conference on FGCS'88, 1988.

[2] Goto,A., et.al., Design and Performance of a Coherent Cache for Parallel Logic Programming Architecture, Proceedings of the 16th Annual International Conference on Computer Architecture, pp. 25-33, 1989.

[3] Kimura,K., et.al., An Abstract KL1 Machine and its Instruction Set, Proceedings of the IEEE Symposium on Logic Programming, pp.468-477, 1987.

[4] Nishida,K., et.al., Evaluation of the Effect of Incremental Garbage Collection by MRB on FGHC Parallel Execution Performance, Technical Report 394, ICOT, 1988.

[5] Ueda,K., Guarded Horn Clauses, Concurrent Prolog: Collected Papers, E.Shapiro, Ed., MIT Press, 1987.

[6] Ohnishi,S., et.al, Evaluation of the KL1 Language System on the Multi-PSI, In Workshop on Parallel Implementation of Languages for Symbolic Computation, University of Oregon, 1990.

[7] Lenoski,D., et.al., Overview and Status of the Stanford DASH Multiprocessor, Proceedings of the International Symposium on Shared Memory Multiprocessing, pp. 102-108, 1991.

[8] Warren,D.H.D., et.al., Data Diffusion Machine - a Scalable Shared Virtual Memory Multiprocessor, Proceedings of International Conference on FGCS'88, 1988.

[9] Scott,S.L., et.al., A Cache Coherence Mechanism for Scalable, Shared-memory Multiprocessors, Proceedings of the International Symposium on Shared Memory Multiprocessing, pp. 49-59, 1991.

[10] Willson,A.W., Hierarchical Cache/Bus Architecture for Shared Memory Multiprocessors, Proceedings of the 14th Annual International Conference on Computer Architecture, pp.244-252, 1987.