ICOT Technical Report: TR-0896

TR-0896

Preliminary Evaluation of a Distributed Implementation of KLIC

by
A. Nakase, K. Rokusawa, T. Fujise
& T. Chikayama

November, 1994

ICOT
Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan
(03)3456-3191~5

Institute for New Generation Computer Technology

Preliminary Evaluation of a Distributed Implementation of KLIC

Akihiko NAKASE, Kazuaki ROKUSAWA, Tetsuro FUJISE, Takashi CHIKAYAMA
Institute for New Generation Computer Technology (ICOT)
21F. Mita-Kokusai Bldg., 1-4-28, Mita, Minato-ku, Tokyo 108, Japan. Phone:+08-3-3456-3193
e-mail: {nakase, rokusawa, fujise, chik}@icot.or.jp

Abstract: KLIC is a portable implementation of concurrent logic programming language KL1. In KLIC, KL1 programs are translated into C programs and compiled into object programs. Through the usage of the C language on a variety of platforms, the sequential KLIC system shows both efficiency and portability. In this paper, we describe the key issues of portable implementation of a distributed KLIC system. Two different configurations of distributed KLIC systems are implemented and evaluated. We propose and evaluate some alternative data transfer mechanisms to maximize parallel efficiency of the distributed KLIC system.

Keywords: Logic programming language, KL1, Parallel processing, Distributed processing, Evaluation

1 Introduction

KL1[1] is a concurrent logic programming language developed by the Japanese Fifth Generation Computer Systems project. KL1 was implemented on parallel inference machines Multi-PSI[2] and PIMs.[3] and contributed to the development of the operating system and many knowledge information application software systems. Although KL1 was efficient on a parallel inference machine, it cannot be used on commercial machines.

KLIC has been developed to make KL1 more widely available. In KLIC, KL1 programs are translated into C programs. Thanks to the high portability of the C language and efficient C compilers, KLIC shows high portability and efficiency.[4]

In order to utilize KLIC systems on commercial parallel machines, we developed a portable configuration for the distributed KLIC system. In this configuration, machine dependent procedures are strictly separated from the distribution library, giving high portability. In our experimental implementation, the parallel efficiency of the distributed KLIC system depends on the efficiency of the platform, e.g. CPU speed and message communication speed. Various message transfer schemes were tried to decrease the number of messages.

In this paper, we describe the portable implementation of a distributed KLIC system, evaluate the parallel execution of benchmark programs, and show some methods for optimizing the message transfer scheme.

Section 2 gives an overview of a distributed KLIC system and its configuration. Section 3 describes the framework of our experimental implementation. Section 4 gives some alternative strategies for a message passing scheme for the distributed KLIC system. Section 5 gives the results of experiments using benchmark programs. Section 6 gives a summary and describes future plans.

2 Overview of the Distributed KLIC System

In the distributed KLIC system, KL1 programs are translated into C programs, compiled into object programs, linked with libraries, and become executable programs. In this procedure, the distribution execution library is

always linked to object programs.

Some copies of these executable programs are forked (or spawned), and these programs execute while communicating with each other. These forked processes are called *KLIC processes*.

In order to maintain the performance of sequential execution, the distributed KLIC system does not change the sequential KLIC core. The distribution execution library is attached to the sequential core as a form of extended library. This mechanism is called a *generic object*[4] in KLIC system.

For the portable implementation, we divided the distribution execution library into a *KL1 level distribution library* and a *machine level distribution library*.

The KL1 level distribution library is directly called from the sequential KLIC core and controls goal migration, distributed unification, distributed garbage collection, and distributed termination detection.

The machine level distribution library is called from the KL1 level distribution library. It initializes KLIC processes and passes actual messages between them.

The configuration of the distributed KLIC system is illustrated in Figure 1.

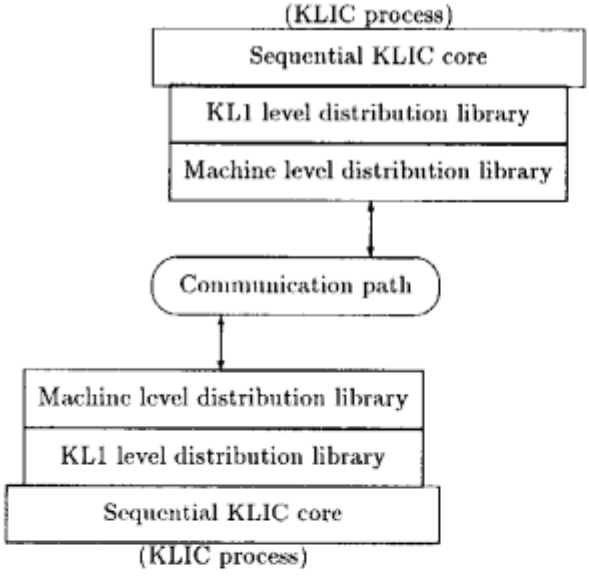

Figure 1: Configuration of the distributed KLIC system

This configuration makes the distributed KLIC system highly portable. It can be ported by changing the machine level distribution library for the target machine architecture and communication path.

2.1 KL1 Level Distribution Library

The KL1 level distribution library controls goal migration, distributed unification, distributed garbage collection, and distributed termination detection. The basic mechanisms of the KL1 level distribution library are the same as for the distributed execution mechanisms of Multi-PSI and PIMs[5].

In the distributed KLIC system, goal migration should be explicitly directed in the source program by pragma `'@node(X)'`. For example,

```
a:-true | b@node(1).
```

means to migrate goal `b` to KLIC process `1`.

In the distributed KLIC system, when a goal is migrated to another KLIC process, the goal is encoded in the *%throw* message and is sent to the other KLIC process.

When a goal is thrown to another KLIC process and the goal contains references to variables, references across KLIC processes appear; these are called *external references*. (Figure 2) We are using the generic object to implement the external references.

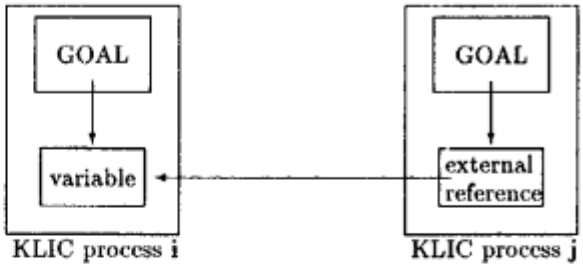

Figure 2: External reference in a distributed KLIC system

When a goal tries to unify some value to an external reference, a *%unify* message is sent to the KLIC process where a variable exists. (Figure 3)

When a goal requires the value referred by an external reference, a *%read* message is send to the referred KLIC process to fetch the value. (Figure 4)

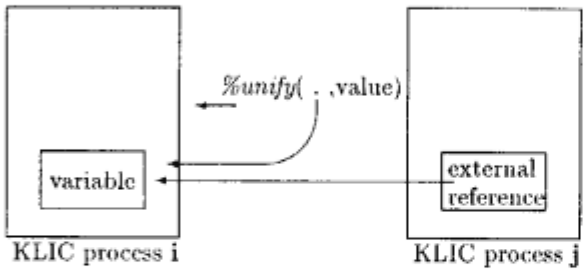

Figure 3: *%Unify* message

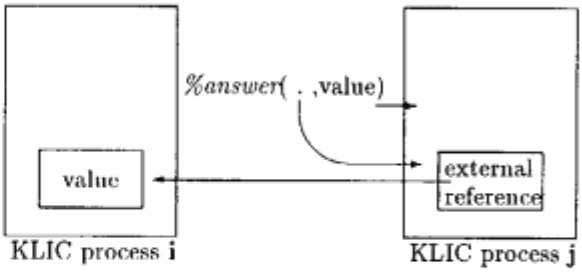

Figure 5: *%Answer* message

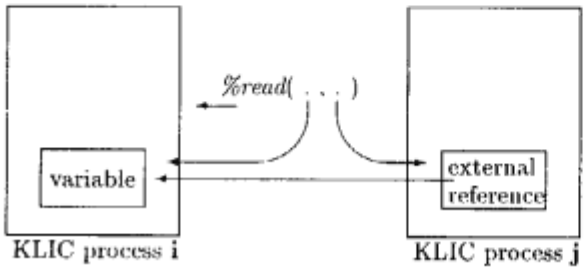

Figure 4: *%Read* message

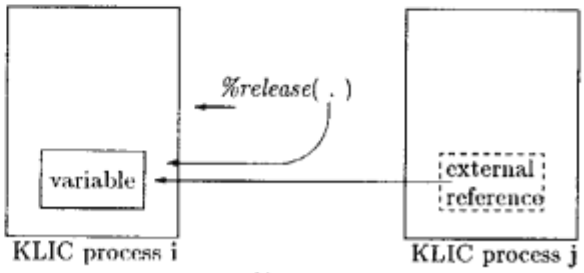

Figure 6: *%Release* message

On receiving the *%read* message, if the variable has been instantiated, an *%answer* message is returned to the KLIC process from where the *%read* message was issued. (Figure 5)

If the variable has not been instantiated yet, the *%answer* message is not sent until the variable is instantiated.

When an external reference is no longer referred to, a *%release* message is send to the referenced KLIC process for garbage collection. (Figure 6)

Thus there are mainly 5 basic messages in the distributed KLIC system, namely the *%throw*, *%unify*, *%read*, *%answer*, and *%release* messages.

2.2 Machine Level Distribution Library

2.2.1 Functions of the Machine Level Distribution Library

The machine level distribution library has two functions. These are: initialization of KLIC processes, and actual message communication between them.

- **Initialization:**
  In initialization, KLIC processes are spawned. The number of KLIC processes is specified when the executable program is invoked. Communication paths are then established between all KLIC processes. Moreover, a console process is invoked, and communication paths are established between the KLIC processes and console process. The console process manipulate the input and output of tracer of KLIC.

- **Message passing:**
  Message passing in the distributed KLIC system is done in a simple manner. Sophisticated message passing primitives, such as broadcast messages or the use of tagged messages, are not necessary.

  Message sending comprises just three actions. These are: creating the send buffer, packing data into the send buffer, and sending the message. Message reception also comprises just three actions. These are: creating receive buffer, receiving message into the receive buffer, and unpacking data from the

receive buffer.

The KLIC process which is to send a message, is called a *message sender process* while the KLIC process which is to receive a message, is called a *message receiver process.*

2.2.2 Communication Paths

The machine level distribution library handles message communication on the communication paths of the platform. We give examples of communication paths.

- **General purpose message-passing library:**
  Recently, many message-passing libraries have been developed, such as PVM[6], and MPI[7]. These libraries have been ported onto many kinds of parallel machines and enhance the portability of distributed KLIC system.
- **Shared memory:**
  In shared-memory machines and virtual shared-memory machines, shared memory can be used for a high performance message path.
- **Machine-dependent communication path:**
  Most parallel machines have their own message passing libraries for maximizing parallel execution performance. If these libraries have the functions described in 2.2.1, they can be used as the machine level distribution library for the distributed KLIC system.

3 Experimental Implementation of the Distributed KLIC System

We used two different communication paths for the experimental implementation of the distributed KLIC system. We used PVM and shared memory for the communication paths.

3.1 Implementation Using PVM

We implemented the experimental distributed KLIC system on PVM.[6] (This implementation is called a *PVM version of the distributed KLIC system.*) PVM is a message passing library running on various kinds of parallel machines, such as networked workstations, shared memory machines, and massively parallel machines.

PVM has library routines which satisfy the functions described in 2.2.1, and we used about 10 of these library routines.

The KLIC process is invoked by *pvm_spawn*. Messages are sent with *pvm_initsend*, *pvm_pk*, and *pvm_send*. Message are received with, *pvm_nrecv* and *pvm_upk*.

3.2 Implementation Using Shared Memory

We also implemented the experimental distributed KLIC system on a shared memory machine. (This implementation is called a *Shared memory version of the distributed KLIC system.*) Figure 7 shows implementation using shared memory.

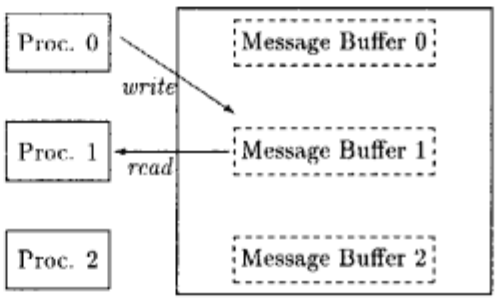

Figure 7: Shared memory version of the distributed KLIC system

Each KLIC process has its own message buffer on shared memory. This message buffer is used for receiving messages. KLIC processes are invoked by fork or *pvm_spawn* (If PVM is available).

The message sender process exclusively writes the message into the message buffer of the message receiver process. Then the message receiver process exclusively read the message from its own message buffer.

3.3 Volume of Source Code

The volume of the source code in machine level distribution library and in the total distributed KLIC system is listed in table 1.

Table 1: Volume of source code in KLIC

	PVM	Shared Memory
Machine level distribution library	480 lines	690 lines
(Initialization)	(330 lines)	(510 lines)
(Message Passing)	(150 lines)	(180 lines)
Sequential core	10000 (C) + 3400 (KL1) lines	
KL1 level distribution library	2100 lines	

The KL1 level distribution library and machine level distribution library are written in C. The sequential core is written in C and KL1.

The source code in the machine level distribution library is less than 5% of the total code for the KLIC system. In this configuration, the distributed KLIC system is easily ported to other architectures. [1]

4 Some Alternative Communication Methods

There are some alternative message communication methods for distributed KLIC system.

4.1 Transfer Methods for Structured Data

We can consider two strategies for transferring structured data.

- **Lazy transfer mode for sending structured data:**
  This strategy sends only one level of the structured data. For example, in transferring list `[1,2,3]`, 1 (`car` of the list) and pointers to `[2,3]`(`cdr` of the list) are sent. `[2,3]`(`cdr` of the list) will be sent when it is really necessary. This prevents unnecessary transfer of elements of the structured data. (Figure 8) This strategy is called *lazy transfer mode.*

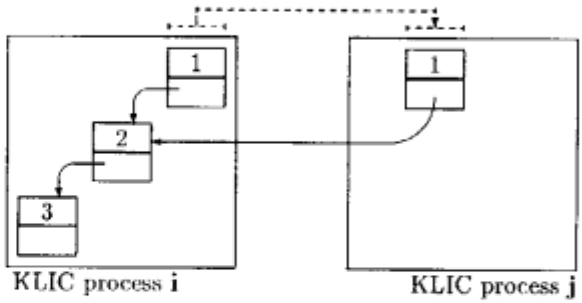

Figure 8: Lazy transfer mode for structured data

- **Eager transfer mode for sending structured data:**
  This strategy sends the whole data structure based on the pointers to the structure. For example, in transferring list `[1,2,3]`, all the list structures are sent. This reduces the number of messages transferred. (Figure 9) This strategy is called *eager transfer mode.*

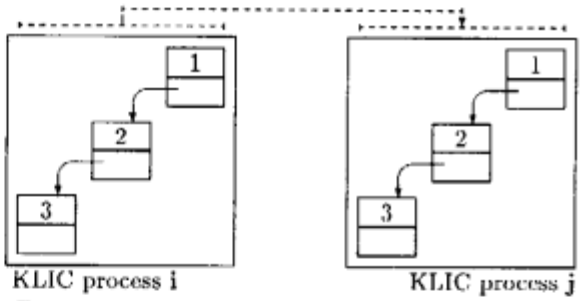

Figure 9: Eager transfer mode for structured data

4.2 Methods to Detect Message Arrival

In a distributed KLIC system, the message receiver process should be able to detect the arrival of a message during computation, and receive the message to allow a quick response.

There are two alternative methods.

- **Sender interrupts receiver:**
  In this strategy, after the message has been sent, message sender process interrupts the message receiver process. This can be achieved by, for example, by sending a signal to the message receiver

[1] In our implementation, we developed the PVM version of distributed KLIC system first. To port it to a shared memory version of the distributed KLIC system takes only one person-day.

process. This strategy allows a quick response.

- **Receiver checks message arrivals:**
  In this strategy, the message receiver process checks for the arrival of messages. For example, if an interval timer is available, each KLIC process sets the interval timer and checks for the arrival of a message synchronized with the timer interrupts. The message receiver process is interrupted less frequently during computation than with the first method.

5 Evaluation of the Experimental Implementation

In this section, we evaluate the experimental implementation.

5.1 Test Environments

5.1.1 Benchmark Programs

- **queen:**
  Counts the number of solutions to 13-queen problems.
- **poly:**
  Polynominal Interpolation program. One process is dedicated to load distribution.
- **genetic analysis:**
  Protein sequence analysis program using multiple alignment. One process is dedicated to load distribution.

5.1.2 Hardware Configuration

Measurements were done on a SparcCenter2000. SparcCenter2000 is a shared-memory multiprocessor machine, which has 40MHz SuperSparcs as its CPUs.

- **PVM version of the distributed KLIC system:**
  We invoked PVM on a SparcCenter2000 and execute the distributed KLIC system. The PVM daemon assigns KLIC processes to physical processors, and communications between KLIC processes are done via the PVM daemon.
- **Shared-memory version of the distributed KLIC system:**
  We used the shared memory of the SparcCenter2000 in just the same way as described in Section 3.2. Invocation and KLIC processes are handled by PVM.

We measured execution time (response time) of the benchmark programs and the number of messages transferred.

In the tables in this section, *Time(sec)* is the execution time of the benchmark program, *Speed Up* means an increase in parallel speed, and *Messages* is the number of messages transferred.

For the measurement of speed up, we executed the benchmark programs with 1, 2, 4, and 8 processors. In `poly` and `genetic analysis` programs, we used 2, 3, 5, and 9 processors. Because in these programs, one process is dedicated to only load distribution.

5.2 Evaluation of Experimental Implementations and Communication Strategies

5.2.1 Transfer Methods of Structured Data

First, we compared the eager transfer mode and the lazy transfer mode for structured data. In this experiment, the message sender process interrupts the message receiver process after sending each message.

Table 2 shows the results of measurements using the PVM version of the distributed KLIC system, and Table 3 shows the results of measurements using the shared-memory version of the distributed KLIC system.

The shared memory version shows a better performance than the PVM version.

This is because, in the PVM version, due to the low performance of the communication path, message communication forms a bottleneck in the parallel performance.

The eager transfer mode shows a better performance than the lazy transfer mode, especially in the PVM version of the distributed KLIC system.

In the eager transfer mode, the number of messages are significantly decreased compared with the lazy transfer mode, and the message communication bottleneck is eased.

In general, if the appropriate message transfer strategy is selected, a reasonable increase in speed can be achieved for the `queen` and `poly` programs.

Table 2: Comparison of transfer mode(PVM version)

Eager transfer mode					
queen	Processors	1	2	4	8
	Time (sec)	176	104	54	33
	Speed Up	1	1.69	3.26	5.33
	Messages	0	330	495	575
poly	Processors	1+1	2+1	4+1	8+1
	Time (sec)	100	78	43	25
	Speed Up	1	1.28	2.33	4.00
	Messages	414	727	880	970
genetic analysis	Processors	1+1	2+1	4+1	8+1
	Time (sec)	64	88	62	52
	Speed Up	1	0.73	1.03	1.23
	Messages	104	3754	3997	4153
Lazy transfer mode					
queen	Processors	1	2	4	8
	Time (sec)	183	103	69	57
	Speed Up	1	1.78	2.65	3.21
	Messages	0	2904	4356	5060
poly	Processors	1+1	2+1	4+1	8+1
	Time (sec)	122	105	74	61
	Speed Up	1	1.16	1.65	2.00
	Messages	4008	4957	5392	5638
genetic analysis	Processors	1+1	2+1	4+1	8+1
	Time (sec)	83	93	76	66
	Speed Up	1	0.89	1.09	1.26
	Messages	1092	6757	6770	6778

Table 3: Comparison of transfer mode(Shared Memory version)

Eager transfer mode					
queen	Processors	1	2	4	8
	Time (sec)	171	84	44	23
	Speed Up	1	2.04	4.89	7.43
	Messages	0	330	495	569
poly	Processors	1+1	2+1	4+1	8+1
	Time (sec)	96	71	36	18
	Speed Up	1	1.35	2.67	5.33
	Messages	414	730	869	958
genetic analysis	Processors	1+1	2+1	4+1	8+1
	Time (sec)	63	70	39	26
	Speed Up	1	0.90	1.61	2.42
	Messages	99	4016	4062	3953
Lazy transfer mode					
queen	Processors	1	2	4	8
	Time (sec)	174	90	50	27
	Speed Up	1	1.93	3.48	6.44
	Messages	0	2904	4356	5060
poly	Processors	1+1	2+1	4+1	8+1
	Time (sec)	96	72	36	20
	Speed Up	1	1.33	2.67	4.80
	Messages	4008	5022	5479	5727
genetic analysis	Processors	1+1	2+1	4+1	8+1
	Time (sec)	71	51	39	31
	Speed Up	1	1.39	1.82	2.29
	Messages	1854	6578	6679	6766

In the `genetic analysis` program, problem division is done very well, but message communications between sub-problems are concentrated in a short period, forming a bottleneck in parallel speed.

This kind of program is not effective for distributed KLIC system on a shared-memory machine, because effectively there is only one message communication path. (In the PVM version of distributed KLIC system, it is a single PVM daemon process. In the shared memory version, it is one shared memory and one memory bus.)

We intend to port the distributed KLIC system and the `genetic analysis` program to network machines and evaluate the performance there.

5.2.2 Methods to Detect Message Arrival

As described in Section 4.2, sending a signal after sending the message is an effective way to get a quick response to the message. But it is disadvantageous to the message receiver process because of the high cost to signal handling.

The objective of this experiment is to examine the trade-off between quick message response and the cost to signal handling.

In this experiment, we used the PVM version of the distributed KLIC system, which sends a signal message after sending each message.

In the shared memory version of the distributed KLIC system, we don't use a signal to indicate message arrival. The message sender process directly accesses the interrupt flag of message receiver process, which is checked every reduction cycle. Therefore, the message receiver process can detect message arrival without overhead.

We then examined two alternatives.

- The message sender process sends a signal to the message receiver process after sending each message.
- The message sender process does not send a signal to the message receiver process. The message receiver process checks for the arrival of messages periodically.

The results are listed in Table 4. We used the eager transfer mode for transferring structured data.

Table 4: Comparison of methods to detect message arrival

Sending a signal for all messages					
queen	Processors	1	2	4	8
	Time (sec)	176	104	54	33
	Speed Up	1	1.69	3.26	5.33
poly	Processors	1+1	2+1	4+1	8+1
	Time (sec)	100	78	43	25
	Speed Up	1	1.28	2.33	4.00
genetic analysis	Processors	1+1	2+1	4+1	8+1
	Time (sec)	64	88	62	52
	Speed Up	1	0.73	1.03	1.23
Sending no signal for any message					
queen	Processors	1	2	4	8
	Time (sec)	184	99	50	27
	Speed Up	1	1.86	3.68	6.81
poly	Processors	1+1	2+1	4+1	8+1
	Time (sec)	99	75	47	20
	Speed Up	1	1.32	2.11	4.95
genetic analysis	Processors	1+1	2+1	4+1	8+1
	Time (sec)	68	84	56	41
	Speed Up	1	0.81	1.21	1.66

In all benchmarks, sending no signal gives a better performance.

In our benchmark programs, the problem is divided uniformly and the size of each subproblem is large. Therefore, it is not necessary to get a quick response to the message, because there are many goals to be computed before the message response arrives.

But, there are some programs which need a quick response to messages. For example, there are some typical stream-based KL1 programs, in which generator processes and consumer processes are linked by a stream and assigned to different KLIC processes.

Thus, selecting the method used to detect message arrival can be effective as long as the charactor and behavior of the program are known.

6 Conclusion

We have described key issues in the portable implementation of distributed KLIC system. Two different configurations of distributed KLIC system are actually implemented and evaluated.

We evaluated the increase in parallel speed of the distributed KLIC system for some benchmark programs and application programs.

Our evaluation shows that the performance of distributed KLIC systems mainly depends on the performance of the message path. But some optimization of the data transfer method can be effective in easing the bottleneck caused by a low performance message path.

We are now currently attempting to port the distributed KLIC system to various kinds of parallel architecture, including a virtual shared memory machine and a massively parallel machine.

References

[1] K. Ueda and T. Chikayama, "Design of the Kernel Language for the Parallel Inference Machine," *The Computer Journal*, Vol.33, No.6, pp.494–500, 1990.

[2] K. Nakajima, Y. Inamura, N. Ichiyoshi, K. Rokusawa, and T. Chikayama, "Distributed Implementation of KL1 on the Multi-PSI/V2," *Proc. International Conference on Logic Programming*, pp.436–451, 1989.

[3] K. Hirata, R. Yamamoto, A. Imai, H. Kawai, K. Hirano, T. Takagi, K. Taki, A. Nakase, and K. Rokusawa, "Parallel and Distributed Implementation of Concurrent Logic Programming Language KL1," *Proc. International Conference on Fifth Generation Computer Systems*, pp.436–459, 1992.

[4] T. Chikayama, T. Fujise, and D. Sekita, "A Portable and Efficient Implementation of KL1," *Proc. International Symposium on Programming Language Implementation and Logic Programming*, 1994, Manuel Hermenegildo and Jean Penjam

eds., Lecture Notes in Computer Science, #884, Springer-Verlag.

[5] K. Rokusawa, A. Nakase, and T. Chikayama, "Distributed Memory Implementation of KLIC," *Workshop on Parallel Logic Programming and its Programming Environments* pp. 151 162, 1994. University of Oregon, CIS-TR-94-04.

[6] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, and V. Sunderam, *PVM 3 USER'S GUIDE AND REFERENCE MANUAL*, TM-12187, Oak Ridge National Laboratory, Tennessee, 1994.

[7] J. J. Dongarra, R. Hempel, A. J. G. Hey, and D. W. Woalker, *A proposal for a userlevel, message passing interface in a distributed memory environment* Technical Report TM-12231, Oak Ridge National Laboratory, February 1993.