並列論理型言語処理系 KLIC における通信の高速化

伊川 雅彦, 大野 和彦, 中島 浩, 富田 眞治

京都大学 工学部
〒 606-01 京都市 左京区 吉田本町

E-mail: {ikawa, ohno, nakasima, tomita}@kuis.kyoto-u.ac.jp

内容梗概

並列論理型言語 KL1 で記述されたプログラムをメッセージ交換型並列計算機上で動作させる場合、頻発する細粒度通信によって大幅に実行速度が低下してしまう。本研究では、KL1 の実行において頻発する細粒度通信によるオーバヘッドを軽減するために、データ送信時に相手の必要とするデータのみを一括して送信する一括送信手法を提案し、本手法を KLIC のランタイム上に実装した。本論では、特に実装方式について詳述する。この一括送信手法によって、無駄なデータを送信することなしに、粒度の高いプロセッサ間通信が実現できる。性能評価の結果、通信回数で 1/3 ~ 1/6 の回数削減、および実行速度で 3 ~ 5 倍の速度向上が達成された。

Improvement of Inter-PE Communications for Parallel Logic Programming Language System KLIC

Masahiko Ikawa, Kazuhiko Ohno, Hiroshi Nakashima, Shinji Tomita

Faculty of Engineering, Kyoto University
Yoshida-hon-machi, Sakyo-ku, Kyoto 606-01 Japan

E-mail: {ikawa, ohno, nakasima, tomita}@kuis.kyoto-u.ac.jp

Abstract

In the execution of parallel logic programming language KL1 on message-passing multiprocessors, fine-grained communications occur frequently. This cause a drastic performance decline. To decrease overhead of communication, we proposed *packing-send* method and implemented this method on KLIC runtime system. This method decreases the number of communications by sending required data in a lump. Using this method, we can attain coarse-grained communications without sending redundant data. As a result of evaluation, we obtained 3 ~ 5 times speedup, and reduced the number of communications to 1/3 ~ 1/7 compared with that of without *packing-send*.

1　はじめに

並列論理型言語 KL1 は、第五世代計算機計画の核言語として開発され、シンタックスの単純さや並列性を自然に記述できることなどの特徴をもつ。そして、現在までに並列推論マシン上の OS や様々な応用プログラムを記述することで高並列処理への有効性を実証して来た。

KLIC では低レベル部分の最適化に関しては、C コンパイラを用いるため比較的効率のよいコードが得られる。しかしながら、頻発する細粒度通信などによるオーバヘッドのため、実行速度の面で手続き型言語に及ばず、実用的な処理系実現のためには実行形式の改良が必要である。そこで本研究ではこのオーバヘッドを減少させることを目的とし、一括送信 [4] により通信回数の削減を行う KLIC のランタイムの拡張を行った。

以下第 2章では背景として並列論理型言語 KL1, その処理系 KLIC について説明し、続いて第 4章で一括送信手法 [4] について説明する。次に第 5章で性能評価の結果を述べ、第 6章でまとめを行う。

2　KL1 と KLIC

本節では、並列論理型言語 KL1[1] 及びその処理系 KLIC [2][5] の概要について述べる。

2.1　並列論理型言語 KL1

2.1.1　プログラム構造

KL1 は Flat GHC に基づく言語で、以下の形をした節の集合で表される。

$$H :- G_1 \cdots, G_m | B_1 \cdots, B_n.$$

H, G, Bは述語であり、それぞれクローズヘッド、ガードゴール、ボディゴールと呼ばれる。述語は '述語名/引数の個数' で表記される。

実行の単位はゴールと呼ばれる。与えられたゴールをヘッドと同一化し、その各ボディゴールを次の新たな実行ゴールとする。この過程を繰り返すことによってプログラムの実行が行われる。このとき、述語 Hを定義する各節のヘッド H_iとの同一化およびガード部 $G_{i1}, \cdots, G_{im}$ の実行を並列に行うことができる。ガード部では引数の具体化を行うことはできない。続いて、ガード部が成功した節のうち一つが選択され、そのボディ部 $B_{i1}, \cdots, B_{in}$ が並列実行される。

ガードゴールに書けるのは組込述語のみで、ボディゴールには、組込述語とユーザ定義述語を記述できる。また、実行開始時の初期ゴールとして、`main/0` が与えられる。ボディゴールはそれぞれが並列に実行されることもある。また実行中のゴールで未具体化変数への参照が行われると、そのゴールの実行は中断し、別のゴールが選択され実行が続けられる。

2.1.2　データ構造

KL1 のデータ構造には、次の 3 種類があり、セルと呼ばれる基本単位で構成される。

- 論理変数: 論理変数は参照ポインタで表現され、特定の型は持たない。具体化によって値へのポインタが格納される。論理変数は一度具体化されると、別の値に再び具体化することはできない。
- アトミックデータ型: シンボリックアトムや整数などがこれにあたり、値がセルに直接格納されている。
- 構造データ型: 構造データは複数のセルからなり、これらのセルには任意の型のデータ (またはデータへのポインタ) を格納することができる。構造データ型の一般的なものとしてはファンクタやリストがある。

2.2　KL1 処理系 KLIC

KLIC は、ICOT で開発された KL1 の処理系であり、KL1 プログラムをいったん C プログラムに変換し、オブジェクトコードの生成はターゲットマシンの C コンパイラが行う方式をとっている。よって、通信関係などの機種依存部を除き移植性の高い処理系となっている。

実行プログラムは KL1 プログラムから得られたオブジェクトコードと KLIC のランタイム

ライブラリをリンクすることで作成される。実行ゴールの選択などのプログラム全体の制御は、カーネルと呼ばれるランタイムルーチンが行う。

2.2.1 データ構造

基本データ構造

KLICでは前述のKL1のデータをセルにタグをつけることによって表現している。

ゴールレコード

実行の単位であるゴールは、ゴールレコードとして表現され、次に実行されるゴールへのポインタ、対応する実行コードへのポインタ、引数からなる。

Generic object

KLICには、KL1の基本データの他にgeneric objectと呼ばれるデータ構造が存在する。generic objectはデータ領域とデータを操作するメソッドと呼ばれる手続きからなる。

カーネルはgeneric objectの存在のみを知っており、操作を定義しているメソッドを必要に応じて呼び出す。また、実際に定義されているデータや操作の内容には関知しない。

後述するKLIC分散処理系に必要なデータ構造やそれに係わる処理はこのgeneric objectを用いて実現されている。これによって、カーネル部を変更することなしに分散環境が実現できる。

generic objectにはdata, consumer, generatorの3つのタイプがあり、用途に応じて使い分けられる。consumer objectとgenerator objectは共に未具体化変数を表すタグの付いたセル(以下REF)に指されている。このような状態を本論ではobjectがREFにフックしているという。consumer objectがフックするREFに対して具体化が行われたとき`UNIFY`メソッドが起動される。またgenerator objectがフックするREFに対して、具体化が行われると`UNIFY`メソッドが、参照が行われると`GENERATE`メソッドが起動される。

2.2.2 KLIC分散処理系

KLIC分散処理系の計算モデルは、局所メモリを持ちネットワークで結合されたノードが、互いにメッセージ通信をしながら処理を行うというものである。自動負荷分散は行われず、負荷分散を行うには明示的にプログラム中に指定しなければならない。$\texttt{goal}(X_1, \cdots, X_n)\texttt{@node}(N_c)$という形のボディゴールがノード$N_p$上に現れると、このゴールはノード$N_c$に送られ、そこで実行される。全てのノードで実行可能なゴールがなくなると、プログラムの実行は終了する。終了の検出はいずれかのノードに割り付けられた荘園と呼ばれるプロセスが行う。

外部参照ポインタ

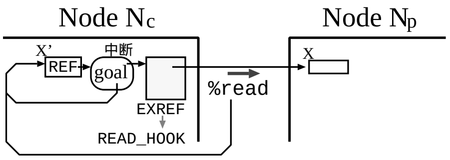
変数や構造データの送信が行われると、ノード間に渡る参照ポインタが生まれる。これを外部参照ポインタという。

外部参照ポインタを経由した参照や具体化にはメッセージ通信という機種に依存した処理を伴う。そこで外部参照ポインタをgeneric object (`EXREF`)を用いて実装し、カーネルの移植性を高めている。

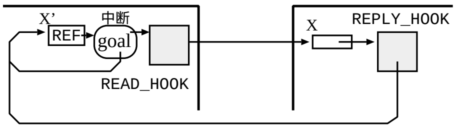
以下では、ゴール`goal(X)`がノードN_pからノードN_cに送られた場合を用いて外部参照ポインタに対する操作の説明を行う。

外部参照ポインタへの参照

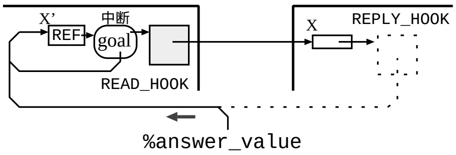
1. N_cからN_pに読みだし要求メッセージ(readメッセージ)が送られる。ゴール`goal(X)`は実行を中断し外部参照ポインタにフックする。このとき、外部参照ポインタはgeneratorからconsumerに変化する。(図1(a))。
2. readメッセージをノードN_pが受信すると、`X`が具体化済であればanswer_valueメッセージによって`X`の具体化された値を返信する。`X`が未具体化変数であれば値の返信は中断され(図1(b))、その後`X`が具体化されると具体値がanswer_valueメッセージにより送られる(図1(c))。



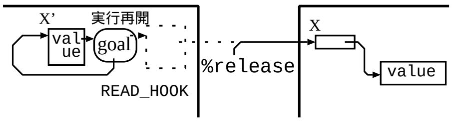
(a)readメッセージの送信



(b)Xの具体化待ち



(c)具体値の返信



(d)外部参照ポインタの解放

図 1: 外部参照ポインタの参照

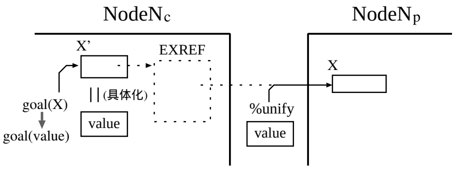


図 2: 外部参照ポインタの具体化

3. answer_value メッセージをノード N_cが受信するとメッセージを同一化し、その結果、中断ゴール `goal` は実行可能ゴールキューにつながれる。そして、ノード間ガーベジコレクションのためにノード N_pに対して release メッセージを送り、`X` が `goal` から参照されなくなったことを知らせる (図 1(d))。

外部参照ポインタの具体化
`EXREF` に対し具体化が行われると、具体値を相手ノードに送信する。これを unify メッセージと呼ぶ。(図 2)。

3 本手法の概要

3.1 KLIC の問題点

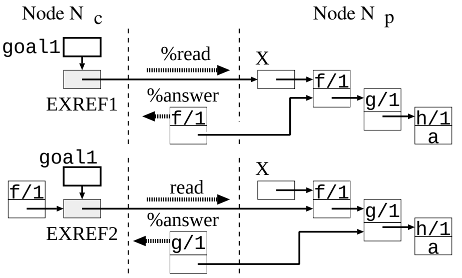
KLIC には KL1 の実行モデルによる速度低下という問題が存在している。KLIC では変数の具体化や参照を行うノードが一般的には実行時まで判明せず、必要となった時点で送信を要求するという要求駆動的なデータのやりとりを行う。このため、粗粒度のメッセージ通信を前提とした並列計算機を対象とした場合、細粒度通信の頻発が大きなオーバヘッドになる。具体的な問題としては次に述べるようなものがある。

1. 第 2.2節で述べたように実体が他ノードにある変数に対し参照が行われた場合、1 回の参照につき 3 回のメッセージ通信が必要であり効率が悪い。

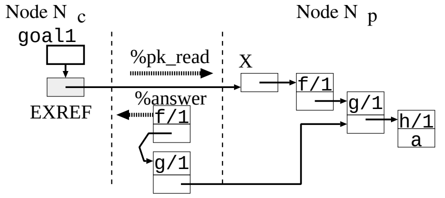
2. 相手が必要としないデータの送信を避けるために、オリジナル KLIC ではある変数 X が構造データに具体化されていた場合には、answer_value メッセージや unify メッセージは構造データの最外側の値しか返信されない。すなわち複雑な構造データを参照する場合にはネスト毎にメッセージ通信が必要になり、メッセージ数が増大する (図 3.a)。

3.2 手法の概要

本手法は外部参照ポインタの参照によって生じる answer_value メッセージおよび、具体化によって生じる unify メッセージが構造データである場合に、相手プロセスの必要とするデータ全てを一括して送信する。これにより不要なメッセージを送信することなしにメッセージ数を減少させることが可能になる。従来の方法では、構造データ `f(g(h(a)))` のうち `f(g(X))` を N_cが必要だとすると、6 回の物理通信が必要であった (図 3.a)。そこで本手法を用いると、必要な物理通信は 2 回ですみ高速化が実現できる (後述する理由により release メッセージも不要)(図 3.b)。



(a)一括送信なし



(b)一括送信あり

図 3: 構造データの送信

4　一括送信手法

KL1 では変数は実行時までそのデータ型は判明しない。しかし、静的解析として、モード解析 [4] を行うことよって、変数の各出現における参照・具体化の別が判明する。さらに、タイプ解析 [4] によって、実行時に取り得るデータ型の集合が決定できる。そこで本手法ではまず、静的解析により得られたこれらの情報を用いて、データ送信時に具体化されているデータのうち、相手の必要とする部分を送信バッファに格納する KL1 コードを作成する。以下、これを一括送信コードと呼ぶ。そして、実行時にこのコードを実行することによって、前述の一括送信を行う。

4.1　静的解析手法

4.1.1　モード解析

[4] で提案されているモード解析を行うことによってゴールの各引数に対し送信モードが定義できる。その定義を以下に示す。

in: 引数が送信側で具体化され、その具体値が構造データである場合。

out: 引数が受信側で具体化され、その具体値が構造データである場合。

normal: 具体化の行われるプロセスが判明しない場合、または具体値がアトミックデータである場合。

4.1.2　タイプ解析

KL1 の変数は型を持たないため、ここでいうタイプとは、ある変数が実行時にどのようなデータ型に具体化され得るかを示したものである。ただし、実行パスによって同じ変数が異なるデータ型に具体化されるプログラムも存在するため、タイプは可能性のある具体値の型の集合で表す。

4.1.3　一括送信ゴール

上記のモード・タイプ解析によって一括送信コードが生成される。この一括送信コードを実行コードとしてもつ KL1 ゴールを一括送信ゴールと呼ぶ。

4.2　一括送信手法の実装方式

以下では、ノード N_p において `goal`$(X_1, \cdots, X_n)$`@node`(N_c) というボディゴールが実行され、ノード N_c にこのゴールが送信された場合を例に、一括送信の実装方式に関して詳述する。

4.2.1　一括送信バッファ

一括送信を行う際の問題点は、一括送信ゴールはメッセージの到着等により、中断される可能性があるため、通常の送信バッファを用いることができないことである。これの解決法として、本実装では一括送信用の送信バッファを 1 つ用意し、これに対して排他制御を行う方法を用いた。この一括送信用の送信バッファを以下では一括送信バッファ(pk_buffer) と呼ぶ。

また、一括送信バッファの排他制御の方法については後述する。

4.2.2　一括送信手法の実装方式

`goal/n` のある引数 `X` について、送信モードにより以下の処理を行う。

(a) 送信モードが in の場合

- in モード外部参照ポインタの生成
  `X` は N_cからの読み出し要求メッセージの受信により、answer_value メッセージとして具体値を一括送信する。そこで、一括送信コードへのポインタを持った外部参照ポインタ (`IN_EXREF`(generator)) を作成する。`IN_EXREF` がフックする REF を `X'` とする。

- 参照値の読みだし
  in モード外部参照ポインタに対してデータの参照が起こった場合の処理について説明する。

1. N_cでの外部参照ポインタへの参照によって `IN_EXREF` の `GENERATE` メソッドが起動し、参照するノード N_pに対して、一括送信コードを持った読みだし要求メッセージ (pk_read メッセージ) を送る。`goal/n` は実行を中断し、`X'` にフックする。

   これ以後、`goal/n` は `X` に対して参照も具体化も行わないため、pk_read メッセージには release メッセージの機能も持たせてやる。こうすることで release メッセージが不要になる。

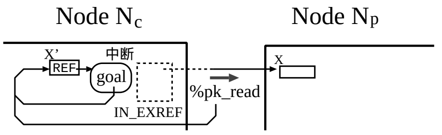
2. pk_read メッセージをノード N_pが受信すると、`X` が未具体化変数であれば `X` に consumer (`PK_REPLY_HOOK`) をフックし、値の返信は中断される (図 4(b))。その後 `X` が具体化されると `PK_REPLY_HOOK` の `UNIFY` メソッドが起動し、次の処理を行う。

   また pk_read メッセージ受信時に `X` が具体化済の場合も、受信処理として次の処理を行う。

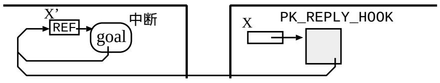
3. 返信処理を行う consumer (`ANSWER_HOOK`) を作り、pk_read メッセージに含まれていた一括送信ゴールを実行可能ゴールキューの先頭につなぐ。一括送信ゴールは `X`,`ANSWER_HOOK` を引数として持ち、次の処理を行う。

4. `X` の具体値をたどりながら、具体値の各要素を一括送信用バッファに格納していく (図 4(c))。最後に `ANSWER_HOOK` の `UNIFY` メソッドを起動するために `ANSWER_HOOK` を適当な具体値で同一化して終了する。

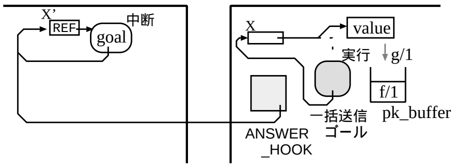
   `ANSWER_HOOK` の `UNIFY` メソッドは一括送信バッファの内容を answer_value メッセージとして、N_c に送信する。(図 4(d))。



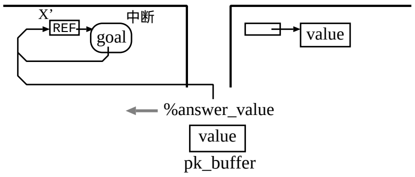
(a)pk_readメッセージの送信



(b)Xの具体化待ち



(c)一括送信ゴールの生成および実行



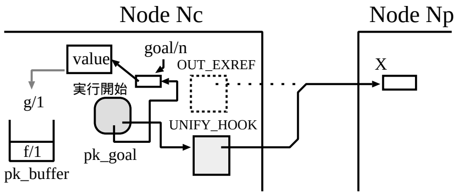
(d)一括送信データの返信

図 4: in モード変数の参照値の読みだし

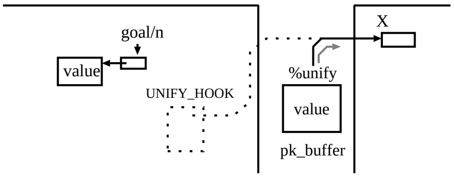
(b) 送信モードが out の場合

- out モード外部参照ポインタの生成
  `X` は N_cの側で具体化されるため、N_cは unify メッセージとして具体値を N_pに一括送信する。そこで、N_c側がゴール goal/n を受信した時、一括送信コードへのポインタを持った外部参照ポインタ (`OUT_EXREF`) を生成する。

- N_cにおける `X` の具体化
  N_cにおいて `X` の具体化が起きると、`OUT_EXREF` の `UNIFY` メソッドが起動する。このメソッドでは返信処理を行う consumer



(a)outモード外部参照ポインタの具体化



(b)一括送信データの送出

図 5: out モード変数の具体化

UNIFY_HOOK を作り、一括送信ゴールを実行可能ゴールキューの先頭につなぐ。

この一括送信ゴールは X の具体値および UNIFY_HOOK を引数として持つ。(図 5(a))。一括送信ゴールが最後に行う UNIFY_HOOK の同一化により、UNIFY_HOOK の UNIFY メソッドが起動し、一括送信バッファの内容を unify メッセージとして、N_cに送信する。(図 5(b))。

(c) 送信モードが normal の場合
従来と同様に 2.2節で述べた外部参照ポインタの処理を行う。

4.3 未具体化変数を含む場合の処理

一括送信ゴールが X の具体値を一括送信バッファに格納する際に、具体値が未具体化変数 Y を含む構造データの場合がある。この場合にもノード間に外部参照ポインタが生じる。このとき、Y の具体化がいずれの側で行われるかが判明していれば、Y の具体値についても一括送信を行うことが望ましい。そこで、次のような処理を行う。

X の具体化と Y の具体化が同じノードで行われる場合は送信モードが in である引数の送信と同様の処理を行う。逆に具体化が行われるノードが異なる場合は、送信モードが out である引数の送信と同様の処理を行う。また Y がいずれのノードで具体化されるか判明しないときは一括送信は不可能であり、Y の具体値がアトミックデータであるときは、一括送信を行う意味がない。よって、従来と同様の処理を行う。

4.4 一括送信バッファの排他制御

ある一括送信ゴール (pk_goal1) の実行中にメッセージ受信等で実行がサスペンドし、この受信処理中に新しい一括送信ゴール (pk_goal2) が生成されることがある。pk_goal1 が終了するまで一括送信バッファの内容は失われてはならないため、一括送信バッファは各一括送信ゴールに対して排他制御を行う必要がある。

この排他制御の仕組みを以下に示す。

1. pk_goal1 は次に生成される一括送信ゴールがサスペンドする時にフックするための変数 (pk_flg1) を引数としてもつ。(本実装では、pk_flg として送信用の consumer (ANSWER_HOOK または UNIFY_HOOK) がフックする変数を用いている)(図 6.(a))
2. pk_goal1 が実行途中で中断され、pk_goal2 が生成されると、pk_goal2 は pk_flg1 にフックしサスペンドする。このとき pk_goal2 は次に生成される一括送信ゴールがサスペンドする時にフックするための変数 (pk_flg2) を引数として持つ。(図 6.(b))
3. pk_goal1 の実行が再開されると、pk_goal1 は終了時に送信用の consumer の UNIFY メソッドを起動させ、かつ pk_goal2 を実行可能ゴールキューにつなぐために pk_flg1 を適当な具体値で具体化する。(図 6.(c))

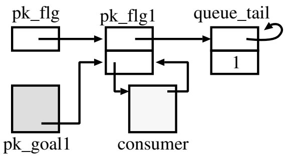
5 性能評価

5.1 評価対象

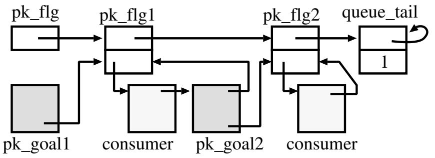
評価用の処理系として、以下のものを用いた。

- PVM 版 KLIC+イーサネットで結合した SparcStation2 台

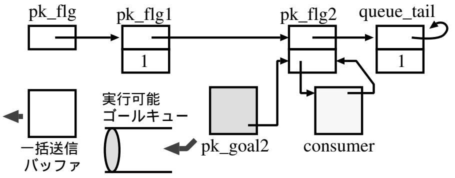
また、評価プログラムとして、次の 3 つのプログラムを用いた



(a) pk_goal1の実行



(b) pk_goal2の生成



(c) pk_goal1の終了

図 6: 一括送信バッファの排他制御

stack 2 ノード間でスタック操作を行うプログラムである。

mastermind 数当てゲームで正解に至るまでの推測の並びを全解探索するプログラムである。

nqueen N-queen 問題の解を全解探索するプログラムである。

5.2 実行結果

上記の 3 プログラムについて、オリジナル KLIC 上および、一括送信コードを用いた拡張版 KLIC 上のそれぞれで実行し、性能評価を行った。計測項目は各セル台数での実行時間、および物理通信回数である。結果を表 1に示す。

一括送信により物理通信回数が 1/3 から 1/7 に減少した。この結果実行速度は 3 ~ 5 倍に向上しており、いずれのプログラムでもよい結果が得られている。

6 おわりに

本論では、KLIC における一括送信による通信の高速化手法と、その性能評価について述べた。

表 1: 各プログラムの実行結果

	一括送信なし		一括送信あり	
セル	時間 (秒)	通信回数	時間 (秒)	通信回数
`stack`				
1	0.07	0	—	—
2	85.8	6504	19.8	1503
`mastermind`				
1	0.23	0	—	—
2	222.7	19327	79.2	6691
`nqueen`				
1	39.8	0	—	—
2	98.3	4733	19.8	671

本研究ではプログラムの静的解析により生成された一括送信コードを、データ送信時に実行することによって、データの一括送信を行うような実装を行った。本手法を用いることによって、余分なデータを送信することなしに粒度の高い通信を実現することができた。

性能評価の結果、3 ~ 5 倍の性能向上が得られ、本手法の有効性が証明できた。

今後の予定としては、より大規模なプログラムでの評価があげられる。

謝辞

日頃御討論頂く富田研究室の諸氏に感謝致します。また、数々の情報と助言を頂いた (財) 新世代コンピュータ開発機構と KLIC タスクグループの方々に感謝致します。

参考文献

[1] Takashi Chikayama, “Introduction to KL1”, August. 1994.

[2] Takashi Chikayama, “KLIC User’s Manual”, October. 1994.

[3] Al Geist,Adam Beguelin,Jack Dongarra, Weicheng Jiang,Robert Manchek, Vaidy Sunderam, “PVM3 User’s guide & reference manual”, August. 1994.

[4] 大野 和彦, 伊川 雅彦, 森 眞一郎, 中島 浩, 富田 眞治, “静的解析による並列論理型言語 KL1 の通信最適化”, JSPP’95,pp.169-176,1995.

[5] 六沢 一昭, 仲瀬 明彦, 近山 隆, 藤瀬 哲朗, “KLIC 分散メモリ処理系の設計と初期評価”, JSPP’95,pp.153-160,1995.