TR-304

密結合マルチプロセッサでの KL1並列処理系の評価

佐藤正俊，後藤厚宏

September, 1987

ICOT
Mita Kokusai Bldg. 21F
4-28 Mita 1 Chome
Minato-ku Tokyo 108 Japan
(03) 456-3191～5
Telex ICOT J32964

Institute for New Generation Computer Technology

密結合マルチプロセッサでのKL1並列処理系の評価

The evaluation of KL1 parallel system on shared memory multi-processor

佐藤 正俊、後藤 厚宏

Masatoshi SATO, Atsuhiro GOTO

(財)新世代コンピュータ技術開発機構

Institute for New Generation Computer Technology (ICOT)

1．はじめに

ＩＣＯＴでは，要素プロセッサが１００台規模の並列推論マシン（ＰＩＭ［1］）を開発中である．ＰＩＭの対象言語は，ＧＨＣ［2］をベースとした並列論理型言語（ＫＬ１）である．ＫＬ１の並列処理方式を決定する上で，通信コストは重要な要因である．このため，ＰＩＭはクラスタを用いて階層化する構成をとる．クラスタ内は，１０台程度の要素プロセッサを共有メモリにより密に結合し，共有メモリを介した高速な通信を可能とする．また，クラスタ間は，各クラスタをネットワークにより疎に結合する．ＫＬ１の処理方式は，クラスタ内の密結合マルチプロセッサ向き処理方式とクラスタ間の疎結合マルチプロセッサ向き処理方式を融合した処理方式となる．これにより，ソフトウェアからみて，プロセッサ台数に対するプロセッサ間の通信コストが段階的に変化するため，ソフトウェアよる負荷分散等の制御が容易になり，全体の処理効率を高められる．

本稿では，クラスタ内の密結合マルチプロセッサ向き処理方式について述べる．クラスタ間の疎結合マルチプロセッサ向き処理方式については，Multi-ＰＳＩシステムとの共通課題として検討を進めている［3］．

密結合マルチプロセッサでのＫＬ１処理方式の利点は，共有メモリを用いてプロセッサ間の通信を低い通信コストで実現できる点にある．しかし，単に必要なデータを全て共有しあう方法では，共有メモリに対するロック操作のオーバヘッドの増加と共有データへのアクセスの集中によるボトルネックが予想される．

一方，ＫＬ１の逐次処理系の評価より，depth-first スケジューリングは，処理コストの大きいサスペンドの数を少なく保てることが知られている．並列プロセッサにおいてdepth-first スケジューリングを行うためには，ゴールのスケジューリング機構をプロセッサ毎に分散するとともに，稼働率が低下しないような高度な負荷分散機構が必要となる［4］．

我々は，これらの課題を検討するために，共有メモリ結合マルチプロセッサBalance 21000 上に言語Ｃを用いてＫＬ１並列処理系を開発し，ＫＬ１処理方式を評価した．本処理系では，アクセス競合の問題を解決するために，ゴールの管理やメモリ領域の割当操作を各プロセッサに分割した．さらに各プロセッサ毎のdepth-first スケジューリングで稼働率を向上させるために負荷分散を要求駆動で行なうＫＬ１処理方式を実装した．

以下では，２章でＫＬ１処理方式の概要を述べ，３章で評価に用いたベンチマークの特性を整理する．４章では，ＫＬ１並列処理系の実行性能，負荷分散方式，要求駆動による負荷分散方式での並列実行時のオーバヘッドについて評価する．

2．ＫＬ１処理方式の概要

2．1．ＫＬ１の処理方式

ＫＬ１の実行はゴールリダクションであり，その実行は以下のようにまとめられる．ここで，ＫＬ１のプログラムは次の形をしたガード付Horn節の集合である．また，ＫＬ１のプログラムはＰＩＭの機械語であるＫＬ１Ｂ［5］コードにコンパイルし実行する．

```
H :- G1, ...Gn | B1, ...Bm. (n>=0,m>=0)
頭部      ガード      ボディ
   受動部              能動部
```

①ゴールの実行制御

ＫＬ１の実行で必要なゴールは，図１のようにゴールレコード（GR）で表現し，ゴールレコード領域に置く．また，ゴール間で共有される変数は，スタックではなくヒープ領域上に置く．ゴールレコードの要素は，ゴールの状態，引数リスト（アトミックデータまたは変数領域へのポインタ），ＫＬ１Ｂコードへのポインタである．また，ゴールの成功／失敗をメタレベルで扱えるように，すべてのゴールレコードはメタコールレコードをノード

としたＡＮＤ木によって管理される．

ゴールレコード領域の中の実行を待つゴールは，レディキューにより管理される．プロセッサは，レディキューからゴールレコードを１つ取り出し，その引数リストをもとにＫＬ１Ｂコードで指定されたゴールリダクションを実行する．

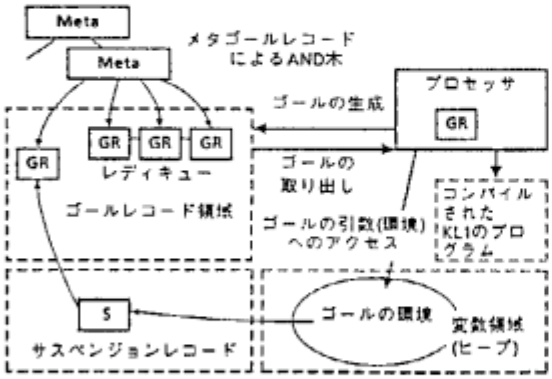

図1．ゴールリダクション

②受動部の実行

複数の候補節から能動部を実行すべき節を１つ選ぶために，ゴールの入力引数に対して，各節の受動部で指定されたユニフィケーションを順次テストしていく．

受動部のユニフィケーションでは，入力引数の指す共有変数の具体化を禁止している．そのような具体化が必要になった場合，プロセッサはその共有変数をスタックし，次の候補節の実行に移る．全候補節をテストしても，節を選択できない場合がある．ここで，入力引数に現われた共有変数が具体化すれば，選択できる可能性がある場合，ゴールリダクションを中断し，その共有変数が具体化されるまで待つ必要がある．そこで，プロセッサは，ゴールの再開時に備えて中断の原因となった（スタックされている）変数とゴールレコードをサスペンドレコードを用いて繋ぐ（bind-hook 方式）．この処理をサスペンド処理と呼ぶ．

③能動部の実行

能動部の実行は，変数の具体化と新ゴールの生成とがある．

変数を具体化したとき，その変数を待っている中断中のゴールがある場合，そのゴールレコードをレディキューに戻す．この処理をリジューム処理と呼ぶ．新ゴールの生成は，新ゴールに必要な引数を新たなゴールレコードに設定し，そのゴールレコードをレディキューに繋ぐ．

２．２．ＫＬ１の並列処理の方針

共有メモリ構成の利点は，共有メモリにゴールリダクションに必要なデータを置き，プロセッサ間通信を共有メモリに対する読み書きの操作で行なうことにより，プロセッサ間通信のコストを小さく抑えることができる点である．このため，ＫＬ１Ｂ命令を共有メモリアクセスでの排他制御（ロック操作）を行うものに拡張する必要がある．しかし，単に必要なデータを全て共有しあう方法では，共有メモリに対するロック操作のオーバヘッドの増加とメモリアクセスの集中によるボトルネック等の問題が生じ，共有メモリ構成の利点を活かした処理効率を得ることが難しくなる．

一方，ＫＬ１の逐次処理系の評価より，depth-first スケジューリングは，処理コストの大きいサスペンドの数を少なく保てることが知られている．

そこで，ＫＬ１の並列処理系では，ゴールリダクションに必要なレディキューを各プロセッサに分散し，さらに各プロセッサのリダクションで使用する領域の割当てをプロセッサ部に行なう（図２）．これにより，ゴールの実行制御，新ゴールの生成時のレディキューに対する排他制御は不要となる．また，各プロセッサは，depth-first スケジューリングが可能となり，その結果サスペンド数を少なく抑えられる．

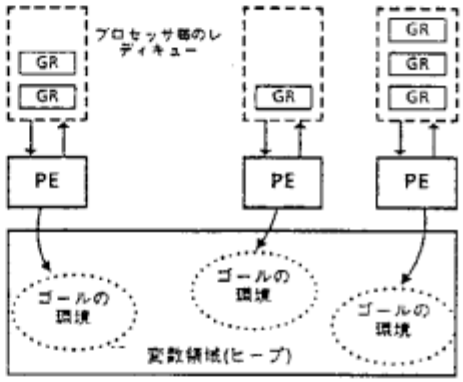

図2　プロセッサ毎のレディキー方式

プロセッサ部にレディキューを分散した場合に必要となる負荷分散方式としては，次のような要求駆動的な方式が有効と思われる．

まず，各プロセッサに個別のレディキューを置き，通常は，depth-first スケジューリングを行なう．負荷分散の操作は，アイドルになったプロセッサからの要求によってのみ起動される．つまり，アイドルになったプロセッサは，忙しいプロセッサに対してゴールの分配を要求する．

ここで，ゴールの要求の仕方，および要求を受けたプロセッサが分配するゴールの選び方が異なる次の２種類の方式を実装した．

①プロセッサがアイドルになったときは，要求フラグと呼ぶ大域フラグをオンにすることで，負荷分散を要求する．分配は，ボディ部の実行で２個以上の新ゴールを生成する時に，分配要求の有無を要求フラグでチェックし，要求有りの場合のみゴールを要求元に送ることで行なう．（以下，rq-l と呼ぶ）．

②プロセッサがアイドルになったときは，レディキュー

の最大のプロセッサを選び，そのプロセッサに対して要求メッセージを送る．分配は，要求メッセージを受け取ったプロセッサが，レディキューよりゴールを取り出してゴールを要求元に送ることで行なう．（以下，rq-aと呼ぶ）．

このような要求駆動的な負荷分散を評価するために，比較対象として次の２種の方式も実装した．

③システムに１つのレディキューを置き，各プロセッサがゴールを取り合う方式（以下，　comaと呼ぶ）．

④各プロセッサに個別のレディキューを置き，各プロセッサがゴールを他のプロセッサにランダムに分配する方式（以下，randと呼ぶ）．

２．３．並列処理系の実装

並列処理系の実装にあたり特に重要であるプロセッサ間通信とロック操作の概要を述べる．

①プロセッサ間通信

前述のプロセッサ間の負荷分散を実現するためには，プロセッサ間のメッセージ通信機構が必要となる．本並列処理系では，各プロセッサ毎にメッセージを受信するポストを設けた．送信プロセッサは，受信プロセッサのポストにメッセージをチェインとして繋ぐ．一方，各プロセッサは，各リダクションの切れ目で自分のポストを調べ，メッセージを取り出す．ここで，メッセージの種類にはゴール要求用とゴール分配用がある．

さらに，負荷分散においては，各プロセッサのレディキュー長などの情報も必要である．これらの情報は，共有メモリの固定領域に各プロセッサが書込み，他のプロセッサが必要時に読み出すこと（共有メモリによる放送）によって通信する．

②ロック操作

本処理系におけるロック操作には，

・リダクション中に起こるヒープ中の変数セルの具体化（書換え）のために行なうロック操作，

・上記のメッセージ通信において，各プロセッサがメッセージを送受信する時に，メッセージのポスト対して行なうロック操作，及び，

・能動部でのゴール生成時のＡＮＤ木の管理のためのロック操作，

がある．

ＫＬ１の単一代入規則により，能動部のユニフィケーションによってのみ，共有変数は具体化される．ただし，受動部でのユニフィケーションにおけるサスペンド処理では，bind-hook 方式によって中断するゴールを変数をフックするため，変数セルの書き換えが生じる．このため，ヒープ領域に対するロック操作は，能動部のユニフィケーションで共有変数を具体化する時と受動部のサスペンド処理時に必要となる．

Balance システムでは，複数のプロセッサから共有メモリへアクセスするときの排他制御を，ＡＬＭ（Atomic Lock Memory ）と呼ばれるtest&setメモリを用いたロック操作で行なう［６］．実際のロック操作は，ＡＬＭ数の制限からシャドーロックと呼ばれるソフトロック方式を使用する．この方式は，実際のロックフラグを共有メモリ上の１バイトで持たせ，その書き換えにアトミック操作を保証しているＡＬＭ上のフラグを用いる．ロックフラグとＡＬＭの対応は，Balance システムが行っている．並列処理系では，シャドーロック方式を用いてロック操作を行なうために，図３のデータ構成とした．ここでタグは，処理系で扱うデータ型を表現し，さらにロックフラグを持つため８バイトで１ワードの構成にしている．

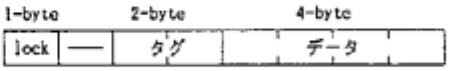

図３．データ構成

３．ベンチマークプログラムの特性

並列処理系の評価に使用したベンチマークを，プログラミングスタイル，大きさ，並列性，計算量の変化の観点から整理する．

３．１．プログラムとプログラミングスタイル

並列処理系の評価に使用したベンチマーク・プログラムは以下のものである．

①８クイーン問題

・ＯＲ並列探索をmerge 述語を用いて，ストリームによるＡＮＤ並列に単純に書き直したもの（8q-m）．

・サスペンドの可能性を排除する等の最適化を行ったもの（8q-s）．

・階層化した再帰的構造（レイヤードストリーム［７］）によるもの（8q-l）．

②構文解析問題（ＢＵＰ）

ＯＲ並列探索を8q-aと同様にストリームによるＡＮＤ並列に直したもの．

③並び換え問題（qsrt：512要素）

④素数生成問題（pria：1000以下の素数）

⑤最大流量問題（maxf　：80ノードに同時に３種類の流量を与える）

各ノードをプロセスとして実現し，プロセス間でメッセージを交換することにより問題を解く．

３．２．プログラムの規模と並列性

プログラムの持つ並列性を，ゴールのリダクション過程を示すゴール木を用いて，並列度と広がり度という２つの尺度によって表現する．

ゴール木とは，入力ゴールを頂点ノード，リダクションによって生成されたゴールを枝ノードとして次々と結んだ物である．まず並列度とは，ゴール木の平均の幅で

あり，

並列度 ＝ 総ノード数 ／ ゴール木の深さ

と定義する．並列度は，breadth-first スケジューリングによる逐次処理系を用いて計測できる．

次に各プロセッサがdepth-first スケジューリングによって処理を進める並列システムについて考える．各プロセッサは，生成されたボディゴールの一つを連続して実行し，残りのゴールをレディキューに戻す．このため，ゴール木の各ノードから２以上の枝があるときが，他のプロセッサに負荷（ゴール）を分散する契機の一つとなる．このため，各ノードの広がり"枝数－１"はプログラムの並列性を示す尺度の一つと考えることができる．例えば，appendのようにボディゴールが１つだけ作り出すものは広がりが０であり，ボディゴールが２つあるものは広がりが１となる．そこで，このようなゴール木の広がり具合を

広がり度 ＝ 平均［ノードの枝数－１］

によって示すことにする．

表１に，各プログラムの総リダクション数（red），並列度と広がり度を示す．ここで，BUP の並列度は，breadth-first 処理系の制限より実行できなかったため示していない．

表１より，プログラムに並列度が充分存在するプログラムは，depth-first スケジューリングにおいても広がり度が大きいことが判る．

表１．並列度と広がり度

	8q-a	8q-k	8q-l	BUP	qsrt	prim	axf
red	108K	39K	19K	36K	8K	17K	40K
並列度	647	563	511	—	14	15	44
広がり度	0.76	0.40	0.64	0.74	0.16	0.01	0.11

表２．breadth-first実行によるサスペンド数

	8q-a	8q-k	8q-l	BUP	qsrt	prim	axf
sp/red	0.28	0	0.13	—	0.37	0.88	0.40

３．３．ゴール間の依存性による計算量の変化

ＫＬ１では，ゴールとして表現された並列プロセス間の同期（依存）関係が変数の待ち合わせとして記述される．このため，処理系のスケジューリングとゴール間の依存関係が合わないと，計算量が増加してしまう．つまり，ゴールの実行に必要な引数が具体化する前にゴールリダクションを試みるとサスペンドが生じるため，余分な命令実行が増加する．

逐次処理系では，多くの場合，depth-firstスケジューリングによって，ゴール間の依存関係に合った処理が可能である．これは，ゴールリダクション間の同期の単位を，各ゴールリダクション毎ではなく，深さ方向に連続した複数のゴールリダクションという大きい単位としていることを示している．

・並列処理系において，プログラムの並列度がプロセッサ台数に比べて十分大きいときは，各プロセッサがdepth-first スケジューリングを行うことによって，逐次処理系と同様の動作をすると期待できる．

一方，プロセッサ台数がさらに多くなり，プログラムの並列度と同等（または以上）になると，各プロセッサは，ゴール間の依存関係に係わりなく，すべてのゴールを並列に処理しようとする．このようなプロセッサ数が大きい並列処理系の動作は，breadth-first スケジューリングを行う逐次処理系の動作によって近似できる場合が多い．そこで，各プログラムにおけるゴール間の依存関係の性質を，breadth-first スケジューリングを行う逐次処理系を用いて調べることにする．

breadth-first スケジューリングでの各プログラムのサスペンドの様子をサスペンド数／リダクション数（sp/red）で表わし，表２に示す．ここで取り上げたaxf 以外のベンチマークは，depth-firstスケジューリングの逐次処理系でサスペンドを起こさない．また，axf は，depth-first スケジューリングで，１７Ｋ回サスペンドを起こし，breadth-first スケジューリングでもほぼ同数のサスペンドを起こす．これは，axf の実行では，他のプログラムに比べて小さい単位でゴール間の同期が必要なためと考えられる．

３．４．プログラムの非決定性

並列処理系の総計算量は，サスペンド処理のためだけでなく，プログラムの持つ非決定性によっても変化する．プログラムの持つ非決定性による変化とは，merge 述語のように引数をＯＲ待ちしている場合，その引数の具体化の順序によって選ばれる候補節が変わることである．

ここで取り上げたベンチマークでプログラムが非決定性を持つものは，8q-a，BUP（merge 述語を使用している）と axf（ＯＲ待ち述語を使用している）である．

４．並列処理系の評価

評価のポイントは，プロセッサ毎にレディキューを分散する方法が有効であるか否かである．この評価のために，２．２節で述べた４種類の負荷分散方式の台数効果を比較し，台数効果の違いの要因の分析を行なう．また，台数効果を妨げている要因の把握が重要であり，その分析を行なう．

４．１．基本性能

並列処理系で，各ベンチマークをプロセッサ１台で実行した時の実行性能は，表３のようになる．表において，実行時間は単位を秒，単位時間（１秒）に行ったリダクションの数をRPS，リダクション当たりのKL1B実行数をKL1B/r，単位時間に行ったKL1Bの命令数をKL1B/sで表わす．ここでの性能は，統計情報収集及びデバック支援の

オーバヘッドが含まれており，その内分けは，統計情報収集が約１０％，デバック支援が約５％である．また，append性能は，１．４ＫＲＰＳである．

表３．プロセッサ単体の実行性能

	8q-a	8q-K	8q-l	BUP	qsrt	prim	mxf
実行時間	125	54	59	52	10	21	147
RPS	974	771	369	729	800	790	271
KL1B/r	10.5	14.4	32.6	15.0	14.0	14.0	38.8
KL1B/s	10K	11K	12K	11K	11K	11K	11K

参考：Balance 21000のＣＰＵはＮＳ３２０３２で，その性能は０．７ＭＩＰＳである．

KL1B命令，ロック操作，プロセッサ間通信処理，サスペンド処理の実行時間は，以下に示すとおりである．

KL1B命令の平均実行時間	約　９０μsec
ロック操作	約　３０μsec
ロック衝突時のオーバヘッド	約　５μsec
メッセージチェイン操作	約２３０μsec
メッセージの解析処理	約２００～３００μsec
サスペンド／リジューム処理	約５００μsec

４．２．負荷分散方式の比較

前述（２．２節）の４つの負荷分散方式のそれぞれを用いてＢＵＰを処理した場合の台数効果を図４に示す．図４より，coss，randは，rq-l，rq-aに比べ，台数効果が３割程度落ちていることが判る．ここで，その要因を知るために，プロセッサ８台での各方式の稼働率，サスペンド数，処理に要したKL1B命令数の比（プロセッサ１台の時を１とする）を調べた（表４）．表４より，cossは稼働率を高く保っているにもかかわらず，サスペンド数が増加し，その結果としてKL1B数が増加したことによって性能を下げていることが判る．また，randは稼働率を高く保てないと同時にサスペンドも増加しており，性能が良くない．

これにより，rq-lやrq-aのように，各プロセッサでdepth-first スケジューリングを行い，さらに，要求駆動によって必要な時だけ負荷分散を行うことが有効であることが判る．

次に，要求駆動に基づく２つの方式，rq-lとrq-aを比較する．両者は，４台程度まではほぼ同じ性能である．しかし，４台以上では，稼働率がほぼ同じであるにもかかわらず，rq-lの性能が劣化している．この理由は，サスペンドの増加によるものである．

２つの要求駆動方式を他のプログラムで比較してみる．表５に，２つ要求方式での8q-Kとmxf プログラム（プロセッサ８台）の稼働率，サスペンド数，台数効果を示す．表より，8q-Kのように並列度が十分有り，サスペンドを起こさないプログラムの場合，２つの要求駆動方式は，ほぼ同じ台数効果を得ている．

一方，mxf のようにゴール木の広がり度の少ないプログラム（表１）の場合，rq-lの方式では負荷分散の契機が少ないため稼働率を保つことができない．一方，rq-a方式では，レディキューにゴールレコードがあれば，それを他のプロセッサに分配できるため，稼働率が高められる．

上記の結果より，要求駆動に基づくrq-aは，稼働率を高く保つと同時にサスペンド数を抑える方式であることが判った．

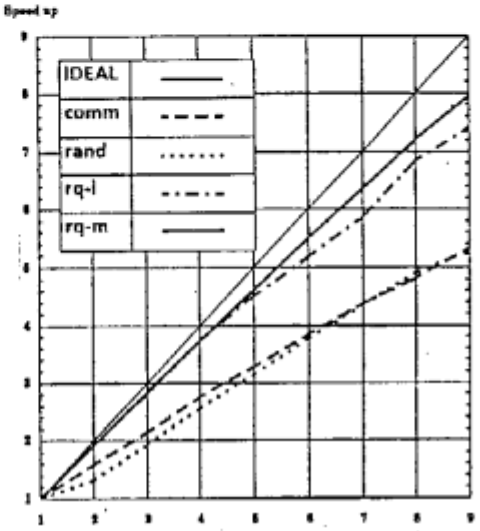

図4.負荷分散方式と台数効果(BUP)

表４．負荷分散方式をサスペンド数（ＢＵＰ）

	coss	rand	rq-l	rq-a
稼働率	99%	94%	96%	96%
サスペンド数	9.3K	6.2K	1.0K	0.5K
KL1B数比	1.27	1.15	0.98	0.98

表５．8q-K，mxf での要求方式の比較

	8q-K		mxf	
	rq-l	rq-a	rq-l	rq-a
稼働率	96%	96%	74%	85%
サスペンド数	0	0	13K	12K
台数効果	7.6	7.5	5.4	6.6

４．３．ベンチマークによる台数効果の違い

各ベンチマークを，負荷分散方式rq-aで実行した場合の台数効果の変化を図５に示す．図５より，プログラムに並列度が充分大きいプログラム（クイーン問題や構文解析問題）は，台数効果が確認できる．また，並列度の小さい場合（並び換えや素数生成，最大流量問題）は，台数効果が飽和する傾向にあることが判る．ただし，mxf では，並列度（表１）がprimやqsrtより大きいにも係わらず，台数効果の伸びが小さい．これは，台数効

果が，ゴール木の並列度だけでなく，ゴール間の同期が単位が大きいか，小さいかによって左右されるためであろう。

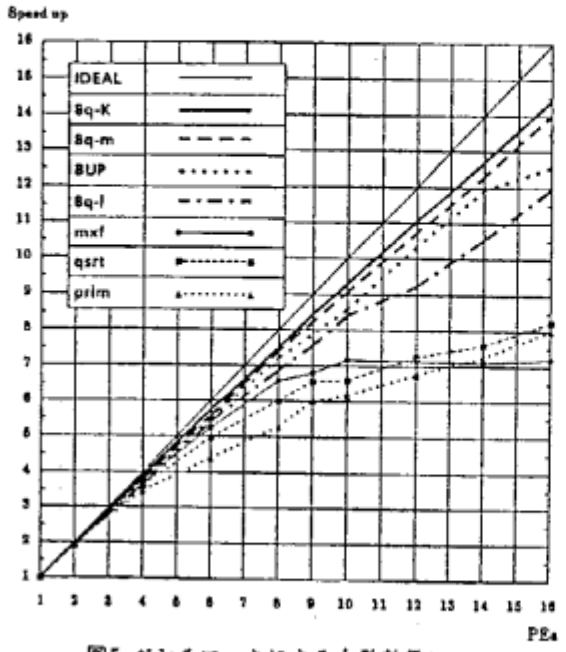

図5. ベンチマークによる台数効果(rq-m)

4．4．台数効果を妨げている要因の分析

並列実行時の台数効果を妨げる要因には，①負荷分散が効果的でなかった場合に生じる稼働率の低下（アイドル率の増加），②ロック操作のオーバヘッド，③プロセッサ間通信のオーバヘッド，④サスペンド数の増加によるKL1B命令数の増加等がある．図6に，上記負荷分散方式rq-m（プロセッサ8台）での，各要因が理想性能（プロセッサ1台の8倍の性能）に対して占める割合を示す．

図6より，稼働率（または，アイドル率）が，台数効果を決める主要因であることが判る．また，ロック操作やプロセッサ間通信のオーバヘッドは，1～5％で比較的少なく，ほぼ一定して存在する．一方，命令数の増加によるオーバヘッドは，プログラムによって大きく異なる．

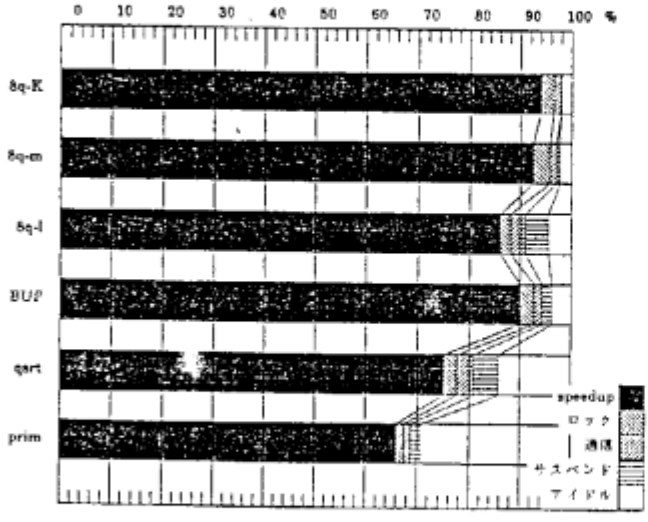

図6. 台数効果を妨げる要因の割合(rq-m)

4.4.1 稼働率

稼働率の低下を招くアイドル時間は，以下の2種類に分類できる．第一は，プログラムに並列性が無いために生じるアイドル時間である．これは，プロセッサがアイドルとなっても分配を受けるべきゴールがなかった時間である．第二は，ゴールの要求からゴールの受け取りまでの待ち時間である．これは，負荷分散方式の実現方法によりその大きさが決る時間である．

図7は，負荷分散方式rq-mにおいて，上記2種類のアイドル率の変化をプロセッサ台数を変化させて得たグラフである．使用したプログラムは，台数効果が異なる3種類のプログラムであり（図5参照），その台数効果は8q-K，BUP，mxf の順に落ちている．ここでは，プログラムに並列性が無いために引き起こされるアイドル時間をidleと呼び，ゴールの要求からゴールの受け取りまでの待ち時間をgwと呼ぶ．図7では，太線の高さはidleとgwの和を示し，破線の高さはgwを示している．

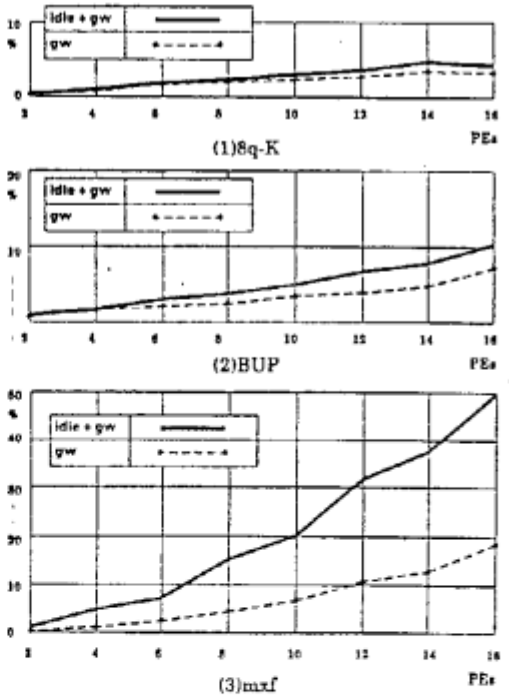

図7. 台数によるアイドル率の変化

図7より，アイドル時間の増加は，idleの増加のためである．idleの増加の理由は，問題の持つ並列性が小さいため，アイドルをなったプロセッサが他のプロセッサよりゴールの分配を受けられないためである．

並列性が充分ある8q-KやBUP では，idleが少ない．ま

た．gwは１６台でも５％前後であり充分小さいと言える．

一方，mxf でidleの増加に伴ってgwが増している．この現出は，分配できるゴールの数が少ない状況では，そのゴールの取り合いが起り，その結果，ゴールの分配待ちが増加してしまうからである．

4.4.2 ロック操作のオーバヘッド

ロック操作のオーバヘッドは，図６で示したように充分小さかった（各プログラムで５％以下）．ここでは，プロセッサ台数を変えた場合のロック操作の特性として，ロック回数（図８）と衝突率（図９）をBq-Kのプログラムの例で示す．ここでの負荷分散方式はrq-aである．ロック操作の特性をさらに詳しく考察するために，ロック操作をその性質から以下の３種類に分類した．

①ヒープへのアクセスのためのロック（H-lock）
　サスペンド時のフック操作と起動部のユニフィケーションでの具体化の時のロック操作

②メッセージ通信のためのロック（S-lock）
　各プロセッサのメッセージチェインへのメッセージの取り出し，つなぎ込み時のロック操作

③ゴール木の管理のためのロック（M-lock）
　ゴールの生成／消滅時のメタコールレコードの更新時のロック操作

図８より，各ロック操作回数は，若干の増加傾向にあるが，ほとんど変化しないことが判る．また，衝突率をみると（図９），H-lock，M-lockは，台数による変化はほとんど無く，S-lockは大幅に増加している．H-lockの衝突率の少ない理由は，ＫＬ１の処理において，ヒープは多重参照が少ない性質によるものである．また，S-lockの衝突の増加の理由は，プロセッサ間通信のためのメッセージ・チェインが，各プロセッサに１つであり，通信のためのアクセスが集中するからである．

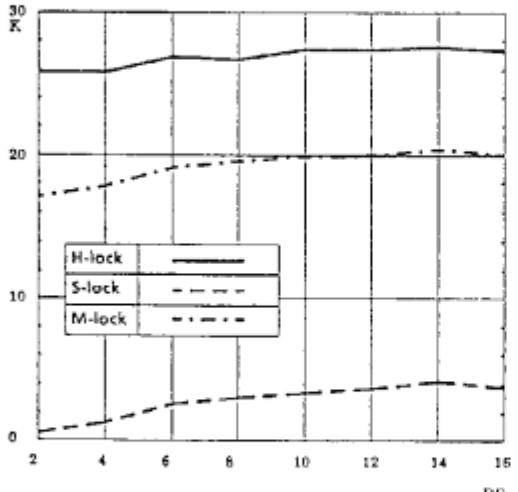

図8. ロック操作回数の変化

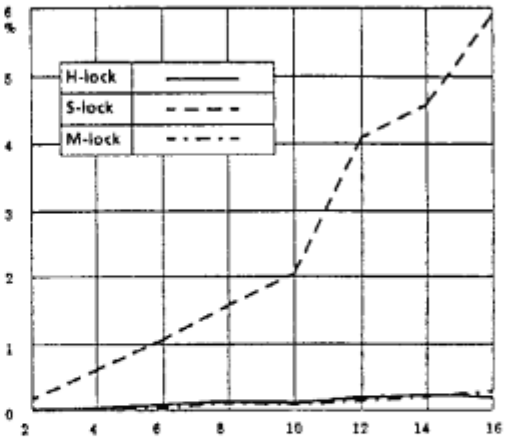

図9. ロック操作の衝突率の変化

4.4.3 プロセッサ間通信のオーバヘッド

プロセッサ間通信のオーバヘッドは，図６で示したように充分小さかった（各プログラムで５％以下）．この理由は，負荷分散を要求駆動で行うことにより，ゴールの分配を抑えているためと通信コストが比較的安いための２つと考えられる．

表６に各プログラムのリダクション当たりのゴール分配率（dist/red）を示す．このデータは，負荷分散rq-aでプロセッサ８台の時のデータである．表より，ゴールの分配は，多い場合でも１０％以下であり充分に小さいことが判る．

表６．ゴール分配率

	Bq-a	Bq-K	Bq-l	BUP	qsrt	prim	mxf
dist/red	2.0%	1.7%	6.7%	1.8%	2.5%	1.0%	5.3%

4.4.4 サスペンドによる命令数の増加

サスペンドによる命令数の増加のオーバヘッドは，３．３節で述べたサスペンドを多く引き起こすプログラムでは，他の要因に比べて大きい（図６）．ここでは，サスペンドによる影響を知るためにサスペンド数（リダクション当たりのサスペンド数：sp/r）とサスペンドが起こった場合のKL1B命令の増加量（KL1B/sp ）を表７に示す．ここでのデータは，負荷分散rq-aでプロセッサ８台で実行した時のデータである．また，サスペンドのためにKL1B命令が増加する理由は，サスペンド処理のための命令の他に，受動部での入力引数のテストと次の候補節を指定するために命令が必要であるためである．これらの命令数は，候補節の数によって決る．

表７より，サスペンド数自身ははそれほど多くないが，サスペンドのために必要な命令数は１ゴールリダクションに必要なKL1B命令数の約１／２必要である．つまり，サスペンドのコスト（bind-book 処理と命令数の増加）はかなり大きい．このため，並列処理することによってサスペンドが増えないようにすることが重要である．

そこで，サスペンド抑止率という尺度を用いて負荷分散方式rq-aで実行した場合の各ベンチマークプログラムの特性を表すことにする．

サスペンド抑止率 ＝ 1 － bf(sp/red)／rq-a(sp/red)

ここで，bf(sp/red)は，breadth-first スケジューリングでのリダクション当たりのサスペンド数を表わし，rq-a(sp/red)は，負荷分散rq-aでのリダクション当たりのサスペンドを表わす．サスペンド抑止率とは，breadth-firstスケジューリングに比べて，どのくらいサスペンドを抑えているかを示す．

また，並列に動作している環境ではサスペンドの発生は非常にクリティカルなタイミングで起る．つまり，現在のサスペンド処理は，全ての候補節のガード部をチェック後，もう一度，真にサスペンドしているかチェックする．しかし，並列環境では再チェックの時点でサスペンド要因の変数が具体化されている場合が起り得る．そこで，rq-aにおいて，サスペンドがどの位クリティカルに回避されているかを，次のようなサスペンド回避率で表すことにする．

サスペンド回避率 ＝ not-sp／sp

ここで，not-spは，再チェック時サスペンド要因の変数が具体化されていた数を表わし，spはサスペンド数を表わす．

表８に，サスペンド抑止率とサスペンド回避率を示す．表８より，負荷分散rq-aは，サスペンドを充分抑止している方式であることが判る．また，プログラムにもよるが，サスペンドが非常にクリティカルなタイミングで発生していることが判る．

表７．サスペンドのオーバヘッド

	8q-a	8q-1	qsrt	prim
sp/red	0.01	0.10	0.08	0.04
KL1B/sp	8	13	7	7

表８．サスペンド抑止率と回避率

	8q-a	8q-1	BUP	qsrt	prim	mxf
抑止率	97%	90%	*	77%	96%	5%
回避率	8%	5%	13%	16%	22%	1%

* BUP は，breadth-first 処理系で実行できないためデータを示せなかった．

5．おわりに

ＫＬ１の並列処理系を実際の密結合マルチプロセッサ上に実装して，密結合マルチプロセッサ向きの並列処理方式及び並列処理系の評価を行った．

この評価により，各プロセッサに個別のレディキューを持たせる方式は，負荷分散を要求駆動でおこなうことで高い稼働率を得られることが判った．各プロセッサが個別にdepth-first スケジューリングすることはサスペンドを減らし高い台数効果を得るために重要であることが判った．また，この処理方式を実際のマルチプロセッサに実装した並列処理系では，台数効果を妨げる要因は稼働率の低下であり，その原因は主にプログラムの並列性小さかったためであった．その他の台数効果を妨げる要因（ロック操作，プロセッサ間通信，サスペンドによる計算量の増加）は，充分小さかった．しかし，サスペンドによるオーバヘッドの影響は予想より大きく，これに対しては，サスペンド処理のコストを軽減する工夫が重要であることが判った．

謝辞

日頃御指導頂くＩＣＯＴ４研，内田俊一室長に感謝します．また，貴重なコメントを頂いたＩＣＯＴ，ＰＩＭグループの方々に感謝します．特に，breadth-first スケジューリングの処理系を提供していただいた宮崎敏彦研究員と最大流量問題（mxf ）を提供していただいた六沢一昭研究員各氏に深く感謝します．

＜参考文献＞

[１] A.Goto et al. Toward a High Performance Parallel Inference Machine - The Intermediate Stage Plan of PIM -. ICOT TR-201.

[２] K.Ueda, Guarded Horn Clauses. ICOT TR-103.

[３] N.Ichiyoshi et al. A Flat GHC Implementation on the Multi-PSI.ICLP'87, 1987.

[４] M.Sato et al. KL1 execution model for PIM Cluster with shared memory. ICOT TR-250.

[５] Y.Kimura et al. An abstract KL1 Machine and its instruction set. SLP '87, 1987.

[６] Balance Guide To Parallel Programming.

[７] 奥村紀，" レイヤードストリームを用いた並列プログラミング" , LPC'87 (ICOT), June 1987