TR-414

GHC処理系における負荷分散方式の検討

安里　彰、岸本　光弘、
小沢　年弘、細井　聡(富士通)

July, 1988

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

ＧＨＣ処理系における負荷分散方式の検討

A Study of Load Balancing Method on GHC

安里　彰　Akira ASATO
岸本　光弘　Mitsuhiro KISHIMOTO
小沢　年弘　Toshihiro OZAWA
越井　聡　Akira KOSHI
林　研司　Kenji HAYASHI
服部　彰　Akira HATTORI

富士通株式会社

FUJITSU LIMITED

1. はじめに

我々は，第五世代コンピュータプロジェクトの一環として，並列推論マシンＰＩＭの開発を行っている。ＰＩＭは１００台規模の要素プロセッサを，共有バス／共有メモリによる密結合されたクラスタとクラスタ間ネットワーク結合の二階層で結ぶ構成のシステムである。ＰＩＭは，並列論理型言語ＧＨＣ〔１〕をベースに設計されたＫＬ１を核言語としている。我々はＰＩＭ上でのＫＬ１の実現に先立ち，共有メモリによる密結合並列計算機である Symmetry 上にＦＧＨＣの処理系を開発した。

処理系作成の第一の目的は，実際に並列動作する処理系を解析して様々な基礎データを採取し，効率の良い並列処理方式を確立するとともに，ＰＩＭ設計へフィードバックすることである。更なる目的としてＧＨＣの言語機能の充実や並列デバッグ法の確立といった，アプリケーションレベルの研究への展開も目指している。

本稿の前半では我々の作成した処理系について概要を述べ，負荷分散の方式について説明する。

並列にプログラムを実行する場合，各プロセッサ間の負荷分散を上手に行うことが重要なポイントとなる。負荷分散の手法はいろいろ考えられるが，それらを全て処理系に組み込んで比較を試みるのは非常に時間を要し，得策ではない。そこで我々は，ＧＨＣの動作に基づいた簡単なモデルを用いて，負荷分散方式の検討を行うことにした。

本稿の後半では，まず上記モデルを概要し，それを用いたシミュレーションの方法を説明する。そして，我々が採用した方式を含む３種類の動的負荷分散方式についてのシミュレーション結果を述べる。更に，静的な情報に基づいて負荷分散を行った場合の性能向上について考察する。

2. 処理系の概要

2.1 ＧＨＣ

ＧＨＣ（Guarded Horn Clauses）は並列論理型プログラミング言語の一つであり，プログラムは次のようなガード付Ｈｏｒｎ節の集合である。

```
H :- G1,...,Gm | B1,...,Bn
```

ヘッド部　ガード部　ボディ部

ＦＧＨＣはガード部に組み込み述語のみを許すという制限をＧＨＣに加えたものである。ＧＨＣの実行はゴールリダクションによって進行する。

2.2 ソフトウェア構成

本処理系は，ＦＧＨＣソースプログラムを中間コード（ＫＬ１の抽象命令であるＫＬ１Ｂ命令）に変換するコンパイラ，アセンブラおよび，中間コードを解釈実行するエミュレータより構成される。エミュレータはＵＮＩＸのプロセスで，複数が同時に動作する。

2.3 処理系の仕様

・ゴール

ゴールはゴール・レコード（ＧＲ）というブロックで表現する。各プロセスはゴール・レコードをキュー管理し，順にリダクション処理を行う。（図１）ゴール・レコードは専用の領域に割り当てる。

・データ

データはタグ付きで表現される。型は，アトム，リスト等の６種類をサポートしている。これらのデータはヒープ領域に割り当てられる。（図１）

・メモリ管理

ヒープ領域，ゴール・レコード領域ともに共有メモリ上にとられるが，両者は別個に管理される。ゴール・レコードはフリーリスト管理を行う。また，これらの領域はプロセス（以後ＰＥと書く）毎に割り当て，競合を抑えている。あるＰＥのフリーリストが尽きると，それを他のＰＥに通知し，フリーなゴール・レコードの再配分を行う。また，コピー方式のＧＣを装備している。ただしこのＧＣは並列に実行するものではなく，各ＰＥが順に自分のルートからのＧＣを行う。

・メタ機能

ゴールの成功，失敗をメタレベルで捉えるように，メタコールをサポートしている。あるメタコールの下のゴールは双方向リンクにより，リング状に結ばれている。

・サスペンション方式

サスペンションは，多重待ちの効率を多少犠牲にしても単一待ちを高速に処理できる方式を工夫した。

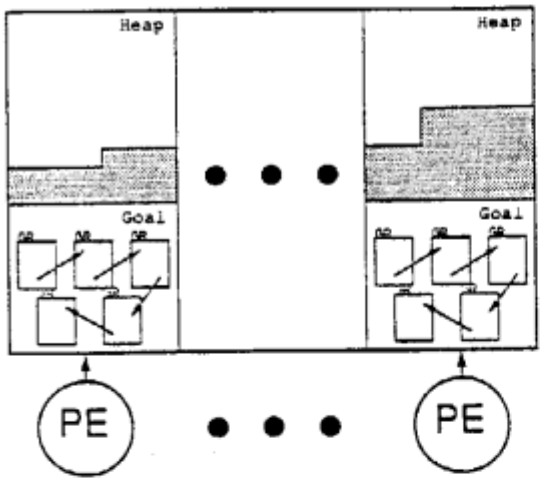


図１　共有メモリ

3.　負荷分散方式

並列システムにおいては負荷分散方式は重要なポイントとなる。負荷分散には動的負荷分散と静的負荷分散がある。前者はプログラム実行中の各プロセッサの負荷に応じて行うもので，後者はプログラム自体に何らかの情報を含ませるものである。我々の処理系では動的負荷分散のみを採用している。以下，我々の方式を説明する。

各ＰＥは独自のスケジューリング・キュー（レディ・キュー）を持ち，そこにはゴールがつながっている。

また，それ以外にエキストラ・キューと呼ぶキューを設けている。エキストラ・キューは全てのＰＥからアクセス可能である。各ＰＥは普段は自分のキューからゴールを得てリダクション処理を行い，生成したゴールは自分のキューにつなぐが，アイドルなＰＥができたり，自分のキューが長くなると，生成したゴールをエキストラ・キューにつなぐようにする。また，アイドルなＰＥはエキストラ・キューからゴールを得る。この方式を用いれば，アイドルなＰＥが出ても，余分なゴールがある限りは速やかにゴールを得られるので，稼働率を高く保つことが期待できる。しかし，アイドルなＰＥが頻繁に現れるとエキストラ・キューへのアクセス競合によって，稼働率が落ちる可能性もある。図２に本方式を示す。

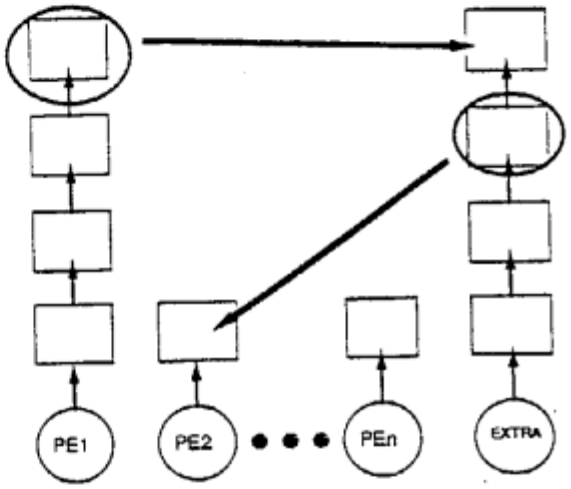


図２　エキストラ・キュー方式

4.　モデルによるシミュレーション

4.1　重み分割モデル

負荷分散には多くの方式が考えられる〔２，３〕が，それらの方式を試験するために処理系自体をインプリメントし直すのは，あまり能率的とは言えない。

そこで我々はＧＨＣの動作に基づく，「重み分割モデル」と呼ぶ簡単なモデルを用いて，負荷分散方式の検討を行うことにした。それは次のようなものである。ゴールに重みという属性を与える。ある重みを持ったゴールはリダクションによって１だけ重みを消費し，いくつかのゴールを生成し，自分は消滅する。新しく生成されたゴール群の重みの合計は，はじめのゴールの重みから１を減じたものになっている。図３に示す。ゴールを何個生成するか，重みをどのように分割するかといった点を工夫することにより，現実のＧＨＣプログラムの振る舞いを模倣することができる。

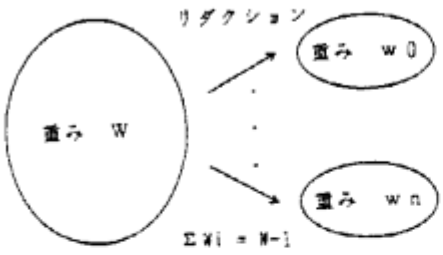


図3　重みの分割

4.2　シミュレーションの方法

本節では上記モデルを用いたシミュレーションについて述べる。複数のＰＥがあり，各ＰＥは自分のキューを持ち，キューにはゴールが連なる。ＰＥはキューの中から１個のゴールを選び，実行（リダクション）する。その結果，新たなゴールが生成され，どこかのキューにつなげられる。選ばれたゴールが必ず実行できる保証はなくサスペンドすることもある。また，キューが空になったＰＥは何らかの手段でゴールを得ようとする。キューは排他制御の対象であり，同時には１個のキューには１個のＰＥしかアクセスできない。

シミュレーションは，適当な初期状態から各ＰＥがゴールの獲得，実行，生成，分散を，ゴールがなくなるまで繰り返す様子を１クロックずつトレースすることで行う。ＰＥの数はパラメタとして可変にする。実行，キューイング，サスペンド等に要するクロック数は我々の処理系での値に基づいて決めた定数を用いる。測定項目は稼働率およびサスペンド回数である。

生成したゴールの分散方法，アイドルなＰＥのゴール獲得方法，実行するゴールの選択方法を規定するのが，負荷分散方式である。その際，各キューの長さや，アイドルなＰＥの数などを基準にするのが，動的負荷分散であり，ゴール自身の持つ情報をもとにするのが，静的負荷分散である。

4.3　プログラムのタイプ

シミュレーションに用いるプログラムのタイプは，実際のＧＨＣプログラムを意識して，典型的な動作をするものとして，ＴＲＥＥ型，ＦＩＬＴＥＲ型の２タイプを採用した。それぞれの動作を説明する。

①　ＴＲＥＥ型

あるゴールは重みがなくならない限り２個の子ゴールを生成する。同じ親ゴールから生成された兄弟ゴール間で優先順位をつけることができる。優先度の高いゴールが実行されるまでは，優先度の低いものは実行できず，もしスケジューリングされればサスペンドする。

親の持つ優先度を子が引き継ぐこともできる。その場合は，親より低い優先度のゴール（子から見れば叔父にあたる）は，子ゴールが全て終了するまで，実行が待たされる。図４のような３通りのパターン（Ａ：兄弟間に優先順位がない場合，Ｂ：兄弟で優先度が違うが親の優先度は継承しない場合，Ｃ：親の優先度を継承する場合）が等確率で起こるようにインプリメントした。

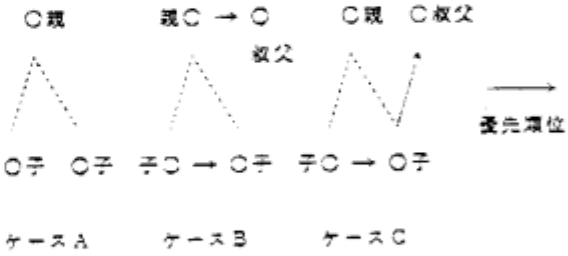


図４　子ゴール生成パターン

②　ＦＩＬＴＥＲ型

図５のように，優先順位の定まった，一定の幅のゴール群を想定する。図で最も左のゴールは無条件で実行可能であるが，それ以外のゴールは自分の左のゴールが終わるまでは実行できない。各ゴールは１個の子ゴールを生成し，優先順位は子供に引き継がれる。幅は３２とした。

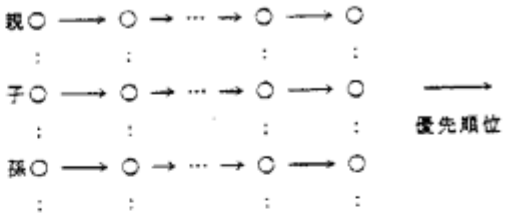


図５　ＦＩＬＴＥＲ型

5.　シミュレーション（１）　―　動的負荷分散

本章では，動的負荷分散方式を変化させてシミュレーションを行った結果を示し，考察を加える。

5.1　各方式の紹介

３通りの方式についてシミュレーションを行った。

①　ＣＱ（Common Queue）

キューが唯一つ存在し，全てのゴールはそこにつながる。各ＰＥはゴールを生成すると，一つは自分が続けて実行するために残し，あとは全てキューイングする。

ゴールがある限り，速やかにスケジュールされるので，高い稼働率が期待できるが，キューへのアクセスが集中すると，アクセス競合が生じやすい。

② ＲＱ（Request ）

〔２〕で提案されていた方式。各ＰＥが自分のキューを持つ。アイドルになったＰＥは，一番キューの長いＰＥにメッセージを送る。メッセージを受けたＰＥは送り手のＰＥのキューに自分のキューからゴールを分け与える。この方式は，自分以外のキューへアクセスする場合でも，メッセージを受けたＰＥに限定されるので，キュー上のアクセス競合が生じないという長所を持つが，アイドルになったＰＥがゴールを得るまでにある程度の時間を要するため，稼働率の点では他方式に一歩譲る。

③ ＥＱ（Extra Queue ）

我々が処理系で採用した方式である。第３章で説明したように，各ＰＥのキュー以外にエキストラ・キューを１本設けてスケジューリングを行う。

5.2 シミュレーションの条件

シミュレーションは，ＰＥの台数を４個から４個刻みで２４個まで変化させ、稼働率とサスペンド数を測定した。リダクションに要したクロック数をｔｒ，全体のクロック数をＴとすれば，稼働率は次の式で与えられる。

稼働率（%） ＝ ｔｒ／Ｔ

また、ＰＥは常にキューの先頭のゴールを選ぶようにした。その他の条件は以下の通りである。

重みの初期値 ＝ ５０００（ＴＲＥＥ）
５００＊３２（ＦＩＬＴＥＲ）

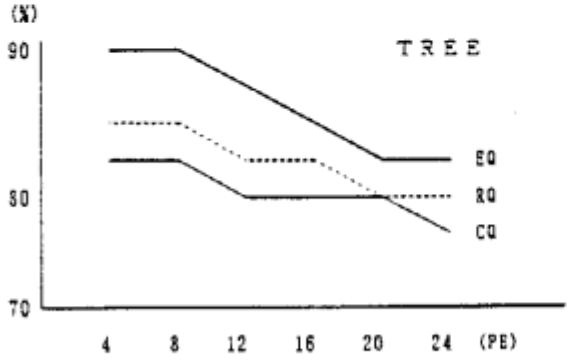
リダクションに要するクロック数 ＝ １０＋１０ｎ
（ｎは生成するゴールの数）

サスペンド処理のクロック数 ＝ １０

キュー操作に要するクロック数 ＝ １

5.3 シミュレーション結果

測定結果を示す。



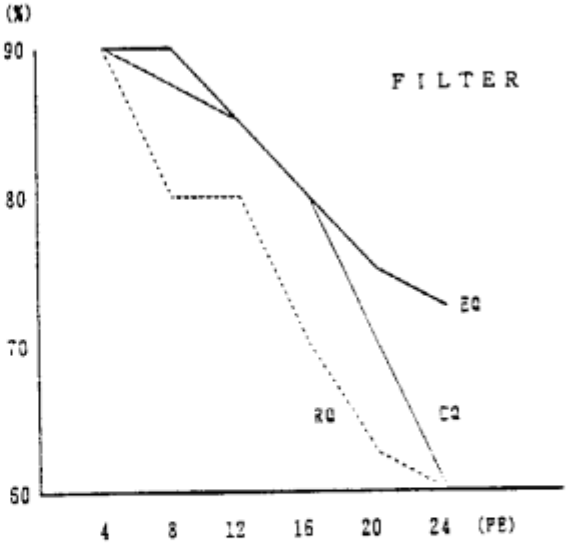


図６ 稼働率（動的負荷分散）

PE	4	8	12	16	20	24
TREE CQ	1035	1148	1204	1242	1185	1235
EQ	704	692	691	723	785	848
RQ	714	714	692	682	700	724
FILTER CQ	386	1295	1814	2461	3542	4893
EQ	510	703	1690	2471	3644	4657
RQ	237	533	1282	1382	1517	1526

表１ サスペンド数（動的負荷分散）

どちらのタイプのプログラムにおいても，稼働率はＥＱが最も優れていることがわかる。また，ＥＱとＲＱのサスペンド数を比べると，ＴＲＥＥではほぼ等しく，ＦＩＬＴＥＲではＲＱの方が少ない。にも関わらずＥＱの稼働率が高いのは，アイドル率の差が効いているからである。ＣＱでは１本のキューへのアクセスが集中するため，本来の稼働率の高さが活かせていないと考えられる。

6. シミュレーション（２） － リジューム優先方式

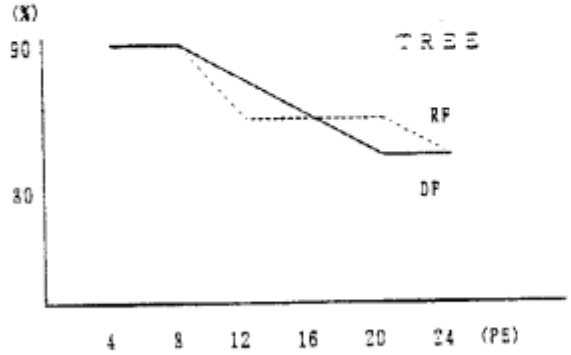
6.1 リジューム優先方式

リジュームされたゴールは絶対実行可能である。

このことから，ゴールを選ぶ基準として，リジュームされたゴールを優先的に選ぶ方式が考えられる。これをＲＦ（Resume First）と名付け，キューの先頭を選ぶ方式ＤＦ（Depth First）との比較を試みた。動的負荷分散方式はＥＱを用い，他の諸条件は前章と同様にした。

6.2　シミュレーション結果

測定結果を示す。



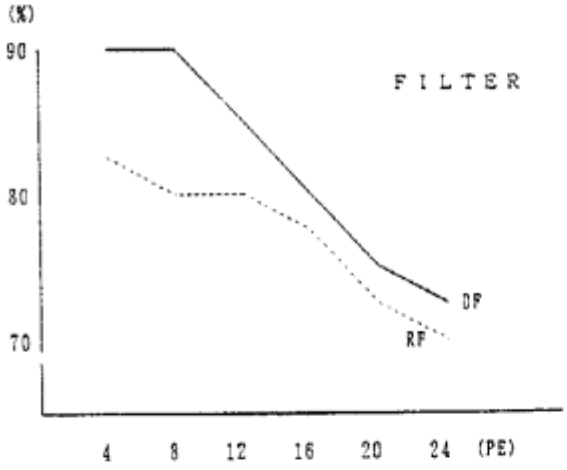


図７　稼働率（リジューム優先方式）

PE	4	8	12	16	20	24
TREE DF	704	692	691	723	785	848
RF	667	706	734	705	756	824
FILTER DF	510	703	1690	2471	3644	4657
RF	6255	5919	5960	6828	7306	7452

表２　サスペンド数（リジューム優先方式）

ＴＲＥＥの場合は稼働率，サスペンド数とも両方式で際立った差異はみられない。ＦＩＬＴＥＲではＲＦの方が，稼働率，サスペンド数の双方で劣っている。結論としては，リジュームしたゴールを優先的にスケジュールしても意味がないと言わざるを得ない。これは，サスペンドしたゴールの子供は親同様にサスペンドすることが多いため，優先的に実行しても仕方ないからだと考えられる。特にＦＩＬＴＥＲ型ではその傾向が強い。

7.　シミュレーション（３）　－　静的負荷分散

我々の処理系では静的な負荷分散は現在行っていないが、処理系の性能を向上させるためには不可欠だと考えている。ここでは，静的負荷分散の目的をサスペンド数の削減に置き，ゴール間の優先度に関する情報を，静的情報と定義する。一般に静的情報を実行以前に得るには、コンパイラの高機能化、ソース上での明示的な記述法の開発などが考えられるが、ここではそういった方法論には触れずに、静的情報がある程度得られた場合の性能向上についてシミュレーションした結果を示す。

7.1　静的情報

①　ＴＲＥＥ型の場合

静的情報として，ゴールに自然数で表す順位属性を持たせた。兄弟間に優先順位が生じた場合，優先度の高い方から０，１，．の順に順位属性を与える。そして，キューの中で順位属性が最小のものをスケジューリングするようにする。また，親の優先度を受け継いだ場合に，順位属性の付け直しを行う方式と行わない方式を区別した。前者をＳ１，後者をＳ２と呼ぶ。

②　ＦＩＬＴＥＲ型の場合

全てのゴールに（ｉ，ｊ）のような２次元座標を振った。ここでｉは図７の左から順に，ｊは上から順に０，１．．．とした。ｉが小さいほど優先度は高い。また，子供を生成する度にｊは増えていく。座標（ｉ０，ｊ０）のゴールは、ｉ０＝０なら無条件で動作可能で、それ以外の時は、（ｉ０－１，ｊ０）のゴールの実行が終わるまではサスペンドする。ゴールを選ぶ際には、ｉ＋ｊが最小なものを選ぶ方式、ｉが最小なものを選ぶ方式を採用した。前者をＳ３，後者をＳ４と呼ぶ。

7.2　シミュレーション条件

シミュレーションは、前節で述べた方式とＤＦを用いた。動的負荷分散方式はＥＱに統一し，その他の測定条件は動的負荷分散の場合と同様にした。

7.3　結果と考察

測定結果を示す。

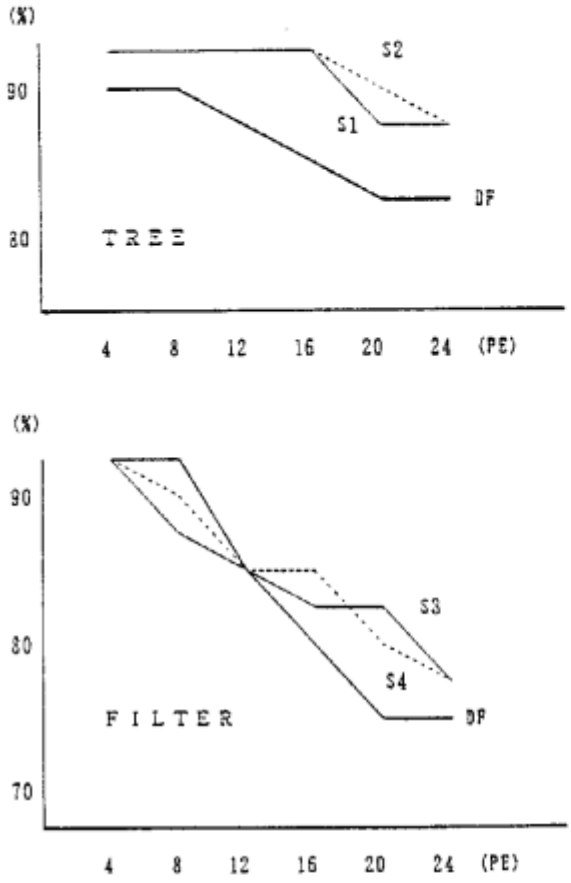


図8　稼働率（静的負荷分散）

PE		4	8	12	16	20	24
TREE	DF	704	692	691	723	785	848
	S1	55	62	48	55	94	103
	S2	26	29	33	54	62	94
FILTER	DF	510	703	1690	2471	3644	4657
	S3	671	1916	2498	3038	3381	5014
	S4	288	958	1225	1712	2692	3431

表3　サスペンド数（静的負荷分散）

ＴＲＥＥでは静的負荷分散によって稼働率が高まり，サスペンドも大幅に抑制されているのがわかる。一方ＦＩＬＴＥＲではあまり効果が上がっていない。またＳ１とＳ２，Ｓ３とＳ４ではＳ１，Ｓ３の方がそれぞれ情報量が多いにも関わらず，むしろ逆効果となっている。ＴＲＥＥ型の場合にＳ２程度の静的情報を取り入れ，ＦＩＬＴＥＲ型では単にDepth First でスケジューリングするのが良いことが分かった。

8.　まとめ

我々の作成したＦＧＨＣ処理系を紹介した。また，負荷分散方式をシミュレーションするためのモデルを提案し，各種シミュレーションの結果を述べた。シミュレーションの結果，次のような知見を得た。

- 動的負荷分散ではＥＱが優れている。ただしサスペンド数はＲＱの方が抑制されている。
- リジューム優先方式は効果がない。
- 静的負荷分散はＴＲＥＥ型プログラムに有効で，静的情報としてはＳ２程度で良い。しかし，ＦＩＬＴＥＲ型プログラムでは静的負荷分散は効果がなく，単なる Depth First方式で良い。

今後も，今回の結果をふまえて，より良い並列システムの研究を進めていくつもりである。

謝辞　日頃御指導下さる林部長，服部室長および研究室諸兄に感謝致します。

参考文献

〔1〕渕他，"並列論理型言語ＧＨＣとその応用"　共立出版

〔2〕M Sato et al, Evaluation of the KL1 Parallel System on a Shared Memory Multiprocessor, IFIP WG 10.3 Working Conference on Parallel Processing in PISA ITALY, April 1988

〔3〕D.L.Eager et al, Adaptive Load Sharing Homogeneous Distributed Systems, IEEE Trans. on Softwear Engineering, MAY 1986