密結合マルチプロセッサ上での FGHC 処理系の実現†

松田秀雄†† 石田英雄†††
金田悠紀夫†† 前川禎男††

本論文では密結合マルチプロセッサシステム Symmetry 上での FGHC 処理系の実現について述べている．FGHC のプログラムをCプログラムの形をした中間コードを介してコンパイルする．中間コードは WAM を FGHC の並列実行用に拡張した中間言語命令から成っている．クイーン，素数生成，クイックソートを実行しそれらの実行時間を測定した．その結果，問題によってサスペンドの発生頻度と回数がかなり異なり，それが台数効果に影響していることがわかった．このことは FGHC では単純に並列度を上げるだけでなく，サスペンドを抑える何らかの方法がないと並列実行の効果が得られにくいことを示している．本処理系ではまた，ヒープのガーベジコレクションを印付け法を並列に行う方式で実現した．これによりガーベジコレクションの処理時間はプロセッサ8台の時の9クイーンで1台の時と比べ約1/5に短縮された．また回収できた領域は9クイーンで約70%，素数生成では99%にものぼった．以上のことから，本処理系のガーベジコレクションの手法が有効であることが示された．

1. はじめに

知識情報処理の応用分野が多様化するにつれて，さらに高速な計算機の必要性が高まってきている．このためには並列処理技術の導入が不可欠と考えられており，知識情報処理の問題を高速に解くための並列計算機の開発が活発に行われている．

並列論理型言語 GHC は言語の実行メカニズムが並列実行を基本としており，解くべき問題の持つ並列性を素直に記述できるようになっている[1],[2]．このため GHC で知識情報処理の応用プログラムを記述する試みがいくつかなされている[3]．しかし，GHC の処理系をいかに実現するか，またそれによってどれくらいの性能が得られるかは，並列計算機上で処理系を実現した例が少ないためさらに研究が必要である．

そこで，本研究では密結合構成のマルチプロセッサシステム Symmetry 上で FGHC の処理系を作成し，その上でいくつかの評価プログラムを実行してその特性を調べた．FGHC は GHC の言語仕様にいくつかの制約を加えて簡単化したものであり，GHC より処理系の実現が容易である．以下では FGHC 処理系の実現方式とその性能評価について述べる．

† Implementing FGHC System on Tight-coupled Multiprocessor by HIDEO MATSUDA (Department of Systems Engineering, Faculty of Engineering, Kobe University), HIDEO ISHIDA (Matsushita Electric Industrial Co., Ltd.), YUKIO KANEDA and SADAO MAEKAWA (Department of Systems Engineering, Faculty of Engineering, Kobe University).
†† 神戸大学工学部システム工学科
††† 松下電器産業(株)

2. FGHC

GHC (Guarded Horn Clauses) は，Concurrent Prolog, PARLOG といった並列論理型言語と同様，コミット選択言語 (Committed Choice Language) であり，プログラムは次のようなガード付き節の有限集合として表される．

$$H :- G1, \cdots, Gn \mid B1, \cdots, Bm. \qquad n, m > 0$$

ここで，"|" をコミット演算子，H と $G1, \cdots, Gn$ および $B1, \cdots, Bm$ をそれぞれヘッド，ガード，ボディと呼ぶ．また，コミット演算子の左側を受動部と呼び，右側を能動部と呼ぶ．

ガード付き節の意味は，宣言的には "," とコミット演算子がともに and とみなされた Horn 節である．手続き的には，"," はその両側のゴールを並列に解くことを表し，コミット演算子は競合する節（同じゴールとのユニフィケーションを行おうとしている節）の中から1つを選ぶということを意味する．

以上のような意味を持つプログラムにおいて，GHC の実行は次のように行われる．

（1） ゴールによる節の1回の呼出しでは，受動部の実行に成功した節が1つだけ選ばれて，その能動部のゴールが並列に実行される．

（2） 受動部の実行では呼出し元の未定義変数を具体化できない．

（3） ゴールの実行する順番については定められておらず，どれから実行してもよい（並列実行が可能）．

（2）の制限により，OR 並列のように1つの変数に複数の値が代入されることはなく，変数の値を保持す

る領域が1変数につき1つですむ（単一環境）.

FGHC (Flat GHC) は GHC のサブセットであり，ガードにユーザ定義の述語が書けないという制約がある．このため，GHC よりさらに処理系の作成が容易になっている．本研究で実現した FGHC 処理系では，実現を簡単にするため上の(3)に対し以下のような制約を加えている．これらの制約は文献 4) など多くの FGHC 処理系でも同様につけられている．

(a)　節の呼出しは上から下に逐次的に行われる．

(b)　受動部の実行は左から右に逐次的に行われる．

これらの制約で，(a) では，受動部の実行が失敗するか，呼出し元の変数への代入が生じた時に，その下の節の呼出しに移る．そして，候補節のいずれも受動部の実行が成功しない時は，その節を呼び出したゴールの実行をサスペンド (suspend) する．サスペンドしたゴールの実行は他のゴールの実行によりその変数の値が決まれば再開でき，これをリジューム (resume) と呼ぶ．また，(b) により受動部は逐次的に実行され，能動部のゴールのみが並列に実行される．

3. 処理系の実現

FGHC 処理系の実現および評価は Sequent 社製の並列計算機 Symmetry S27 の上で行った．Symmetry S27 は 32 ビットマイクロプロセッサ i80386 を要素プロセッサとして最大 10 台までバスにより結合できる[5]．共有メモリを持ち，プログラムとデータは OS を含めてすべてこの共有メモリに置かれるなど典型的な密結合形態を取っている．共有メモリのアクセス競合を抑えるため，要素プロセッサそれぞれに Copy-Back 方式のキャッシュ（容量 64 KB）が付けられている．

Symmetry の OS は DYNIX と呼ばれ UNIX に並列処理機能を加えて拡張したものである．DYNIX は UNIX のプロセス単位での並列処理機能に加え，プロセス間で共有するメモリ領域の割当て，相互排除のためのロック，バリア（プロセス間での一斉同期）といった並列実行制御の機能を提供する．

3.1 コンパイル方式

本研究で実現した FGHC 処理系ではコンパイル方式で実行を行う．コンパイラは途中のフェーズでCプログラムを中間コードとして生成する．中間コードのコンパイルには DYNIX のCコンパイラをそのまま利用し，FGHC からCへのコンパイラを yacc を使って作成した．

```
ヘッド1 :- ガード1 | ボディ1.
            :
ヘッドn :- ガードn | ボディn.
```

(a) FGHC のプログラム（ヘッドの述語が全て同じもの）

```
述語名 ()
{
    if (op (ヘッド1))
    if (op (ガード1))
    {
    op (ボディ1);
    return;
    }
      :
      :
    if (op (ヘッドn))
    if (op (ガードn))
    {
    op (ボディn);
    return;
    }
    suspend ();
}
```

(b) 中間コード（op()はヘッド，ガードまたはボディがコンパイルされた中間言語命令を示す）

図 1　中間コードの形式
Fig. 1 Format of intermediate code.

FGHC のプログラムがどのように C プログラムに変換されるかを図1に示す．FGHC プログラムは定義（同じ述語記号で引数の数も同じ述語をヘッドに持つ節の集合）ごとにまとめられ，1つの定義が1つのCの関数にコンパイルされる．定義内の各節は図1に示すように単純な if 文の連鎖の形にコンパイルされる．これは2章で述べたように，節の呼出しおよび受動部の実行順序をそれぞれ上から下，左から右に固定したためである．

if 文の条件には，ユニフィケーションまたは組込み述語の実行を行う中間言語命令がくる．これらの命令はその実行が成功すれば真，失敗またはサスペンドすれば偽の値を返すので，失敗またはサスペンドの時は次の節の受動部の実行に移る．

すべての節の実行が失敗またはサスペンドした時には suspend( ) が実行される．この命令で失敗かサスペンドの判断をし，サスペンドの時にはサスペンド処理（後述）が行われる．なお，本処理系では1つの定義中の節がすべて失敗した時の処理は厳密には行っておらず，単にそれを検出するだけにとどめている．

能動部がコンパイルされた中間言語命令は if 文の実行部に置かれ，最後は return 文で終わる．定義の呼出しはCの関数呼出しで実現されているので戻り番地がスタックに積まれるが，この情報は使われない．能動部が true の（つまりゴールがない）節が呼び出された時，それまでにスタックに積まれた戻り番地

の情報が return 文により一度に解放される.

3.2 中間言語命令

中間言語命令は ICOT の KL 1[6] と同様, WAM (Warren Abstract Machine) を基礎とし, 次のように FGHC の実行用に拡張した命令を備えている.

- wait 命令：受動部のユニフィケーション命令
- get 命令：能動部のユニフィケーション命令
- cregoal, enqgoal 命令：並列実行の命令
- suspend 命令：サスペンドを行う命令

KL 1 と違うのは各命令がCの関数となっており1命令のレベルが高く, 特にリストの取扱いでは1命令でリストとその2つの要素を処理できるようにしている点である. すなわち, リストを扱う命令には, 第1, 第2要素がそれぞれ未定義変数, 束縛変数, 定数の時の3通りを考えて合計9種類がある. 例えば,

```
p([1]) :-true|…
```

という節は,

```
if (wlcoco(arg, INT, 1, ATOM, 0))
{
:
}
```

とコンパイルされる. ここで, wlcoco( ) はリストの第1要素, 第2要素ともに定数のとき生成される wait 命令であり, この命令の5つの引数はそれぞれ,

arg：ゴール引数の値セルを指すポインタ
INT, 1：整数を表すタグとその値
ATOM, 0：アトムを表すタグとその番号

を表している. 最後の2つはリストの第2要素が nil であることを示す. リスト以外の構造体, ベクタなどを取り扱う命令は今のところ設けていない.

cregoal, enqgoal 命令は能動部のコンパイルで生成される. これらの命令ではゴールレコード (ゴールの実行に必要な情報を集めたもの) をレディキューに登録する. レディキューは1本を全プロセスで共有し, そこへの登録は, 能動部の後ろのゴールから順に行われる. ただし, 1番前のゴールだけは逐次的に実行する命令にコンパイルされる. 例えば,

```
p(X) :-true|q(Y), r(X).
```

という節は,

```
{
cregoal( );
putval(garg, arg);
enqgoal(r 1, 1);
putvar(arg, arg);
execute(q 1, 1);
return;
}
```

とコンパイルされる. ここで, garg はゴールレコード中のゴールの引数を指すポインタであり, enqgoal ( ), execute ( ) の第2引数はゴールの引数の個数を示す.

組込み述語としては整数の四則演算および比較演算を行うものがあるが, これらはそれぞれ対応する組込み述語命令にコンパイルされる. 入出力を行う組込み述語は今のところ用意されていない.

コンパイル時の最適化については, ゴール引数および一時変数の間で不要な転送命令の生成を抑えるにとどめている. 例えば,

```
append([V|X], Y, Z) :-true|Z=[V|Z 1],
                          append(X, Y, Z 1).
```

という節を単純にコンパイルするとゴール引数と一時変数との間の wait 命令と put 命令が生成されるが, 最適化により, この節は,

```
if (wlvrvr(arg, xarg, arg))
{
glvlvr(arg+2, xarg, arg+2);
execute(append 3, 3);
return;
}
```

とコンパイルされる. ここで xarg は一時変数の領域 (ゴール引数領域とは独立している) を指すポインタである. リストとのユニフィケーション命令である wlvrvr (要素がすべて未定義変数の wait-list 命令) と glvlvr (第1要素が既参照の変数で第2要素が未定義変数の get-list 命令) 以外の不要な転送命令は最適化により削られる.

3.3 並列実行方式

本処理系での並列実行の手順を以下に述べる.

まず, 実行に先だってプロセスを生成する. ここで, プロセスとは DYNIX の (つまり UNIX の) プロセスを指す. すなわち, 本処理系でプロセスとは PE の実行を代表するものであり, 実行させたい PE の台数だけあらかじめ生成され, FGHC プログラムの実行全体を通してその個数は変化しない. 以下, 本論文では, プロセスという語をこの意味で使う.

次に, 初期ゴールのゴールレコードを作成しレディキューに登録する. このゴールレコードはプロセスによって取り出され実行される. このゴールが節の呼出

しを行うと，その能動部のゴールのゴールレコードが次々にレディキューに登録され並列実行が始まる．

3.4 スケジューリング方式

前で述べたように，FGHC プログラムの並列実行は，

（1） ゴールレコードのプロセスへの割当て

（2） プロセスの PE への割当て

という2段階で行われる．(1)は，実行時に新たなゴールレコードを要求したプロセスから順に割当てが行われる．割り当てられるゴールレコードの選択には，幅優先 (breadth-first) と深さ優先 (depth-first) の2つの方式がある．これらは，レディキューからのゴールレコードの登録，取出しを，それぞれ FIFO（先入れ先出し）と LIFO（後入れ先出し）の順で行う．後で述べる性能評価のところで2つの方式を比較する．

(2)については，スケジューラがプロセスを共有ランキューに登録し，それを PE に割り当てるという形でスケジューリングが行われる[5]．これは DYNIX により自動的に行われる．

3.5 サスペンド処理

サスペンドの処理は並列実行下で共有変数の束縛状況をみて行わなければならない．ゴールがサスペンドするのには以下のような状況が考えられる．

（1） サスペンドの原因となった変数（以下，サスペンド変数と呼ぶ）が1つだけの時

（2） サスペンド変数が複数あり，それらのうちどちらかの値が決まればゴールの実行が可能な時

（3） サスペンド変数が複数あり，それらのうち両方の値が決まらなければゴールの実行が不可能な時

（4） (2)と(3)とがともに生じた時

ここで，(1)の取扱いは簡単であるが，(2)と(3)の区別およびそれらの混ざった(4)の取扱いは容易ではない．本処理系では簡単のため，(2)と(3)とを区別せずに取り扱っている．すなわち，サスペンド変数が複数個ある時はすべて(2)であると考え，サスペンド変数へ値が代入されるたびにリジュームを行うようにしている．この方法だと(3)または(4)の場合はリジュームの後，すぐにまたサスペンドするが，これを繰り返すことによっていずれ(1)または(2)の状態になる．この方法は本来しなくてもよいサスペンドとリジュームを行う分だけ効率が悪いがサスペンド処理が単純になる．

3.6 データ構造と領域

ゴールレコードの構造と他の領域との関係を図2に示す．これらの領域はすべてのプロセスで共有される．ゴールレコードはコード領域を指すポインタ，引数の個数，引数のブロックを指すポインタから成る．引数中の未定義変数，リストはそれぞれ REF, LIST というタグのついたポインタで表現され，未定義変数の値セル，リストセルはヒープにとられる．引数ブロック，ヒープの管理はフリーリスト（未使用領域の断片のチェーン）によって行う．ヒープの未使用領域は最初は単一の領域であるが，ガーベジコレクションによって複数の未使用領域が断片的に回収される．

サスペンドレコードの構造と他の領域との関係を図3に示す．サスペンド変数は特殊なタグ HOOK がついており他の変数と区別できる．サスペンド変数とサスペンド

p(a) :- true | q(X, Y), r(0, X, Y), s([a, b]).

図 2 ゴールレコードの構造
Fig. 2 Structure of goal records.

p(a) :- true | q(X, Y), r(0, X, Y), s([a, b]).

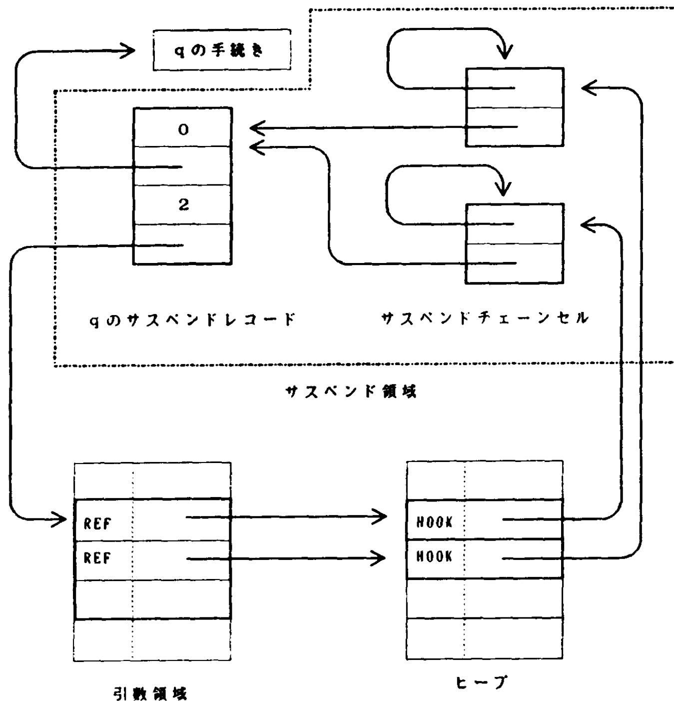


図 3 サスペンドレコードの構造
Fig. 3 Structure of suspended records.

レコードの関係は複雑であり,

(a) 複数のサスペンドレコードが1つのサスペンド変数でつながっている,

(b) 複数のサスペンド変数から1つのサスペンドレコードが指されている,

(c) およびこれらの混ざった状態を表す必要がある.

(a)はサスペンド変数から直接サスペンドレコードを指すのではなく，サスペンドチェーンセルでつながれたサスペンドレコードのチェーンを指すことにより表現している. サスペンドチェーンセルは, 他のセルを指すポインタ，サスペンドレコードを指すポインタの2つから成る. サスペンド変数に1つしかサスペンドレコードがつながっていない時は, セルの1つめのポインタは図3のように自分自身を指す.

(b)はサスペンドレコードの先頭にリジュームのフラグをつけることにより解決している. リジュームが行われるとこのフラグが立つ. これにより, 同一のレコードで何回もリジュームすることが避けられる.

(c)は(a)と(b)との表現を併用することで表せる. なお, 使用中のすべてのサスペンドレコードは, 次のガーベジコレクション処理のために別のチェーンで結ばれてサスペンドリストを構成している.

3.7 ガーベジコレクション

GHC に限らず論理型言語は一般に記号処理に使われるためガーベジコレクションの機能が必要である. しかも, GHC の場合はこれに加えて単一の変数束縛環境を並列に実行するゴールで共有するため, Prolog で行われているような TRO (終端再帰の最適化) による変数領域の解放が困難である. しかし何らかの領域回収を行わない限りヒープの使用領域は実行が進むにつれて単調に増大してしまう.

並列実行時のガーベジコレクションとしては MRB[7] などの手法が提案されているが, この方法だと, 変数参照の度にビット操作を行うなど実行時のオーバヘッドが大きいため, 専用ハードウェアの支援を必要とする. また, MRB では再使用可能領域がすべて回収できるわけではない. これらの理由から, 本処理系では印付け法を並列に処理するガーベジコレクションを行うことにした. 印付け法ではガーベジコレクションの間, ゴールを実行できないが, 並列処理により処理時間が短縮されるので, 全体の実行時間に占めるガーベジコレクション処理時間を抑えることができる.

ガーベジコレクションの処理手順を以下に示す.

① ゴールを実行中のプロセスは, ヒープの未使用領域の大きさがあらかじめ決められた下限値以下になったことを認識すると GC-flag をセットする.

② プロセスは節の最後の処理 (中間言語命令では proceed, execute, suspend) を実行する度に GC-flag を調べ, もしそれがセットされていれば③に進む. そうでなければゴールの実行を続ける.

③ 全プロセスで同期を取る.

④ 1つのプロセスが, GC-flag をリセットし, ⑥の操作で使うポインタの設定などの初期設定を行う.

⑤ 全プロセスで同期を取る.

⑥ 各プロセスは, 自分が実行中のゴール引数からのヒープ参照箇所に印を付ける. 次に, 全プロセスでレディキュー内のゴールレコードを並列に走査していき (各プロセスがゴールレコードを取り合う), そこからヒープを参照している箇所に印を付けていく. そ

の後，サスペンドリストからのヒープ参照箇所についても同様の処理を行う．

⑦ 全プロセスで同期を取る．

⑧ ヒープをプロセスの数に分割し，各プロセスが自分に割り当てられた領域で印付けされていないセルを集め，部分的なフリーリストを構成する．

⑨ 全プロセスで同期を取る．

⑩ 1つのプロセスが，⑧で作った部分的なフリーリストを1つにまとめてフリーリストを構成する．

⑪ 全プロセスで同期を取る．この同期終了をもってガーベジコレクション処理の終了とし，ゴールの実行に戻る．

③，⑤，⑦，⑨，⑪の同期を取る操作は，DYNIX のバリア機能を使って実現している．また，④と⑩の処理は，あらかじめ決められたプロセスによって逐次的に実行される．

4. 性能評価

4.1 実行速度と台数効果

本処理系の性能評価をするため，クイーン（8クイーンと9クイーン），素数生成（512以下と2000以下の素数），クイックソート（要素数が512と2000）の3つの問題を実行した．クイーン問題のプログラムはOR並列探索をストリームを用いてAND並列に変換し，さらに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（文献8）の8Q-K）である．素数生成，クイックソートでは文献9）のプログラムを使った．なお，クイックソートで，ソートすべきリスト要素は別のゴールで発生させた乱数をストリームを用いた通信により与えるようにしている．

各問題のPE 1台と8台での実行時間および台数効果（1台の実行時間を8台の実行時間で割ったもの）を**表1**に示す．本処理系ではいずれの問題でも 4 KRPS (kilo reductions per second) 以上の性能を得ている．表1で台数効果はスケジューリング方式と問題によってそれぞれ違った値となった．以下，これらについて考察する．

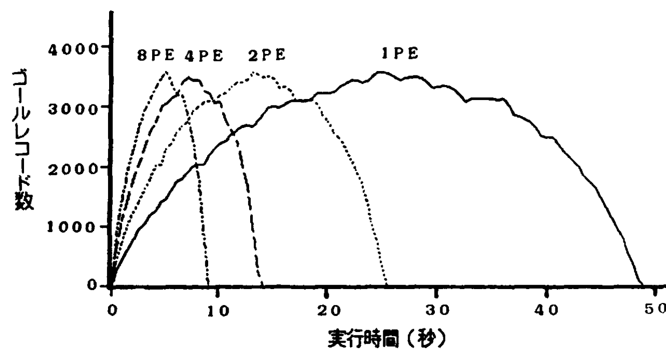
4.2 スケジューリング方式による違い

台数効果は，クイーンとクイックソートの

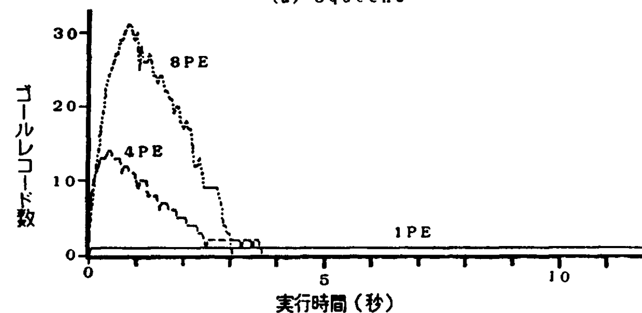
表1 評価用プログラムの実行時間
Table 1 Execution times of evaluation programs.

スケジューリング方式		クイーン 8Q	クイーン 9Q	素数生成 512	素数生成 2000	クイックソート 512	クイックソート 2000
幅優先	PE 1台	9.72	44.53	1.28	11.07	1.72	8.00
	8台	1.72	7.48	0.37	2.42	0.50	2.15
	台数効果	5.7	6.0	3.5	4.6	3.4	3.7
深さ優先	PE 1台	9.55	43.72	1.32	11.17	1.67	7.95
	8台	1.67	7.18	0.35	2.10	0.52	2.13
	台数効果	5.7	6.1	3.8	5.3	3.2	3.7

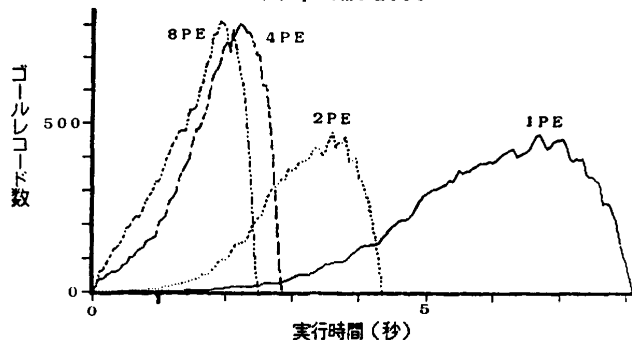
（実行時間の単位は秒）



(a) 9queens



(b) prime 2000



(c) qsort 2000

図4 ゴールレコード数の時間変化
Fig. 4 Transitions of numbers of goal records.

問題ではそれほど差がないが，素数成生問題では深さ優先の方がよい結果が出ている．この原因について調べた結果，スケジューリング方式の違いによってサスペンド回数が大きく違うことがわかった．サスペンド回数は PE 台数が増えるごとに増大する．PE 8台の時のサスペンド回数は，幅優先方式では要素数512の時1361，2000の時5378となるのに対して，深さ優先方式では要素数512の時799，2000の時1130であり，深さ優先の方が幅優先よりもサスペンド回数を抑えられた．

4.3 問題による違い

クイーン問題では比較的よい台数効果が得られ，また問題の規模が増すとさらに台数効果が向上している．素数生成問題では，規模の小さいものでは頭打ちになるものの規模が大きくなるにつれて改善が見られる．しかし，クイックソートでは規模を大きくしても台数効果はそれほど向上しない．

これらの問題の並列度を調べるため，レディキュー内のゴールレコード数の時間変化を調べた（図4）．これは幅優先のスケジューリングでの値である．深さ優先でも同じような傾向が見られた．クイーンが最もゴールレコード数が多いが，素数生成とクイックソートではクイックソートの方がゴールレコード数が多い．つまり図4では，クイックソートでの台数効果が素数生成より低いことを説明できない．

そこで，サスペンドレコード数の時間変化を同様に調べた（図5，スケジューリング方式は図4と同じく幅優先）．図5では，クイーン問題はサスペンドが起こらなかったので省いている．それ以外の2つの問題では，PE 台数が4台をこえると急激にサスペンドが発生するようになる．

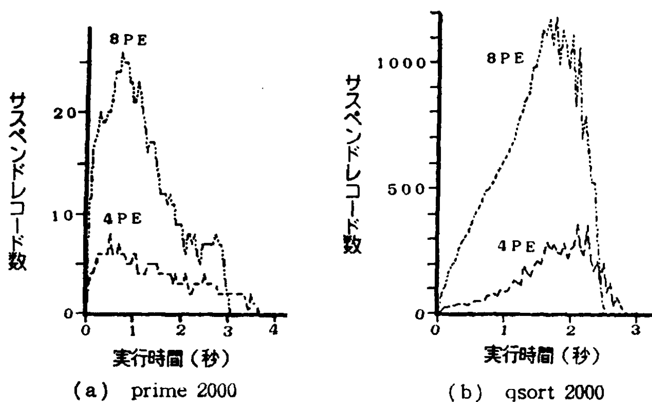


図5 サスペンドレコード数の時間変化
Fig. 5 Transitions of numbers of suspended records.

図4と図5とを比べるとわかるように，素数生成ではサスペンドレコード数はゴールドレコード数を上回ることはないのに対して，クイックソートでは PE 台数が多くなるにつれて急激に増え，8台ではサスペンドレコード数の最大値はゴールレコード数と比べ約1.5倍にもなっている．このことは，クイックソートは PE 台数を増やすとサスペンドが頻発し，サスペンド処理時間が増大することを表している．

これら2つは，ともにストリームを介して通信しながら並列実行を行うプログラムであるが，素数生成の方はストリームに対するフィルタの形で解を求め，クイックソートは分割統合型 (divide and conquer) で，1本のストリームからのデータを1つずつ，2つのゴールのどちらかに分配する．前述のように，データは乱数で与えているので分配はほぼ均等に行われるが，データを送るゴールと受ける2つのゴールは対等にスケジューリングされるので，2つのうち一方は待たなければいけないことになる．以上のことからクイックソートではサスペンドが頻発するものと考えられる．

4.4 ロックの影響

本処理系では，レディキュー，サスペンドリスト，ヒープはいずれも，1つを全プロセスで共有している．したがって，これらの参照・更新では相互排除のためのロック操作を必要とする．ロックによる待ち時間を計測したところ，レディキューのロックによる待ち時間は小さくほぼ無視できることがわかった．しかし，サスペンドが PE 台数の増加とともに頻発する素数生成とクイックソートでは，サスペンドリストのロックによる待ち時間が PE 台数の増加とともに大きくなる．前述のサスペンドの頻発による台数効果の減少は，この待ち時間の増大が原因の1つとなっている．

また，すべての問題でヒープのロックによる待ち時間は PE 台数が増えると増大し，並列度の大きいクイーンでは特に大きな値になった（8クイーンを PE 8台で実行した時の累積待ち時間を1プロセス当りで平均すると約0.3秒）．サスペンドの発生しないクイーン問題で，台数に比例した実行速度の向上が見られないのはこのヒープのロックでの待ち時間が主な要因と考えられる．

台数効果に影響を与える要因としては，このほかに負荷分散方式やアーキテクチャの物理的構造（バス要求の待ち時間やキャッシュのヒット率）など様々なものがあり，これらの影響も

表 2　ガーベジコレクションの処理時間
Table 2　Execution times for garbage collection.

スケジューリング方式		9クイーン 1回目	9クイーン 2回目	2000以下の素数 1回目
幅優先	PE 1台	1.78	1.92	0.65
	8台	0.33	0.33	0.25
	台数効果	5.4	5.8	2.6
深さ優先	PE 1台	0.72	0.77	0.65
	8台	0.12	0.17	0.17
	台数効果	6.0	4.5	3.8

(実行時間の単位は秒)

調べる必要がある.

4.5 ガーベジコレクションの処理時間

ヒープとして 80K 語の領域を取って実行を行ったところ, 9クイーンでは2回, 2000 以下の素数生成では1回ガーベジコレクションが行われた. その処理時間, 台数効果を表2に示す (スケジューリング方式は幅優先). 台数効果は, 9クイーンでは実行速度の台数効果に近い値が得られたが, 素数生成では低く抑えられている. この原因は, 素数生成問題では実行時のゴールレコード, サスペンドレコードの数が少なく (PE 8台の時, それぞれ 18 と7), 3.7 節で述べたガーベジコレクション処理での並列実行できる部分が少ないためと考えられる.

ヒープの回収率については, 9クイーンで1回目, 2回目ともスケジューリング方式, PE 台数によらず約 70%, 素数生成では実に 99%もの領域が回収できた. 回収率は問題の性質によって異なるが, 少なくともこの2つの問題においては非常に多くの領域が回収できることが示された.

5. おわりに

本論文では密結合マルチプロセッサシステム上での FGHC 処理系の実現について, FGHC のプログラムをCプログラムの形をした中間コードにコンパイルする方法を示した.

本処理系の性能評価を行うため, クイーン, 素数生成, クイックソートの3つの問題のプログラムを実行し, それらの実行時間を測定した. その結果, 9クイーンではプロセッサ8台で1台の時と比べ6倍の台数効果が得られた. スケジューリング方式を幅優先と深さ優先とで並列実行を行ったところ, 実行速度に大きな差は見られなかったものの, 素数生成問題ではやや深さ優先方式の方がよい結果が得られた. この原因としてはサスペンド回数の違いが考えられる. また, 問題による台数効果の違いとサスペンド回数との関係について調べ, サスペンド回数が台数効果に影響を与えていることを示した.

さらに, ヒープのガーベジコレクションを印付け法により並列に行う方式を提案し, クイーンと素数生成の問題においてその処理時間, 台数効果, 回収率を計測した. 提案した方式では並列度の高い問題ほど大きな台数効果が得られた. 並列度の低い問題では, ガーベジコレクション処理のうちの逐次実行部分の影響により, 台数効果はあまり得られない. このような場合では, 印付け法ではなくリファレンス・カウントなどでリアルタイム・ガーベジコレクションをすることも考慮する必要がある. ガーベジコレクションで回収できた領域はクイーンで約 70%, 素数生成では 99%にものぼった. 以上のことから, 本方式の有効性が示された.

謝辞　FGHC について資料および有益な御意見を頂いた ICOT および PIM ワーキンググループのメンバーの方々に深謝いたします.

参考文献

1) Ueda, K.: Guarded Horn Clauses, *Proc. of the Logic Programming Conf. '85.9.3*, pp. 225-236 (1985).
2) 上田和紀：並列プログラミング言語, 情報処理, Vol. 27, No. 9, pp. 995-1004 (1986).
3) 渕　一博監修, 古川康一, 溝口文雄編：並列論理型言語 GHC とその応用, 共立出版, 東京 (1987).
4) 大原有理ほか：FGHC ソフトウェアシミュレータの試作, *Proc. of the Logic Programming Conf. '86.7.3*, pp. 93-101 (1986).
5) 平尾延夫：汎用並列計算機 Symmetry/Balance, *bit*, Vol. 21, No. 4 (3月号臨時増刊), pp. 235-243 (1989).
6) Kimura, Y. and Chikayama, T.: An Abstract KL1 Machine and Its Instruction Set, *Proc. of Symp. Logic Programming* (1987).
7) Chikayama, T. and Kimura, Y.: Multiple Reference Management in Flat GHC, Logic Programming, *Proc. Int. Conf. Logic Programming*, Vol. 1, pp. 276-293, MIT Press (1987).
8) 佐藤正俊, 清水　肇, 後藤厚宏：KL1 の並列処理―密結合マルチプロセッサでの並列処理系の評価―, 第 35 回情報処理学会全国大会論文集, 2Q-2, pp. 701-702 (1987).
9) 上田和紀：GHC プログラミング入門, Logic Prog. Conf. チュートリアル資料 (1986).

(昭和 63 年 6 月 20 日受付)
(平成元年 7 月 18 日採録)

松田 秀雄（正会員）

昭和34年生．昭和57年神戸大学理学部物理学科卒業．昭和59年同大学院工学研究科システム工学専攻（修士課程）修了．昭和62年同大学院自然科学研究科システム科学専攻（博士課程）修了．現在，神戸大学助手（システム工学科）．学術博士．論理型言語による並列処理などの研究に従事．日本ソフトウェア科学会，人工知能学会，IEEE, ACM 各会員．ICOT PIM ワーキンググループ委員．

石田 英雄（正会員）

昭和38年生．昭和61年神戸大学工学部システム工学科卒業．昭和63年同大学院工学研究科システム工学専攻（修士課程）修了．現在，松下電器産業(株)に勤務．

金田悠紀夫（正会員）

昭和15年生．昭和39年神戸大学工学部電気工学科卒業．昭和41年神戸大学大学院電気工学専攻修士課程修了．昭和41年電気試験所(現電総研)入所．電子計算機研究に従事．昭和51年神戸大学工学部システム工学科，現教授．工学博士．コンピュータシステムのハードウェア，ソフトウェアの研究に従事．高級言語マシン，並列マシン，AI に興味を持っている．

前川 禎男（正会員）

昭和6年生．昭和29年大阪大学工学部通信工学科卒業．昭和34年同大学院工学研究科博士課程修了．大阪大学助手を経て，昭和36年神戸大学助教授（電気工学科）．昭和47年同大学教授（電子工学科）．現在，同大学システム工学科勤務．システム情報講座担当．この間，システム理論，高級言語マシン，人工知能などの研究に従事．工学博士．電子情報通信学会，電気学会，計測自動制御学会，人工知能学会などの正会員．