計算機アーキテクチャ 83－8
（1990．7．18）

GHCプログラムの設計支援に関する一考察

本城 哲[1]　中島俊介[1]　長谷川晴朗[2]　長谷川隆三[3]

[1]沖通信システム(株)　[2]沖電気工業(株)　[3](財)新世代コンピュータ技術開発機構

並列論理型言語GHC(Guarded Horn Clauses)で記述したプログラムにおける検証及び性能評価を行う手法を提案する．本論文は，GHCプログラムの設計支援法として，プログラム実行前のバグ検出と，プログラム実行後の動作情報の解析を示すものである．まず，第一の方法として，ペトリネットを用いたプログラム検証法と並列度の算出法を示す．次に，もう一つの方法として，トレース情報を利用した負荷の算出法とそのグラフィカルな提示法を示す．

A STUDY ON DESIGN SUPPORT FOR GHC PROGRAM

Akira Honjo[1]　Shunsuke Nakajima[1]　Haruo Hasegawa[2]　Ryuzo Hasegawa[3]

[1]Oki Telecommunication Systems Co., Ltd.　[2]Oki Electric Industry Co., Ltd.
[3]Institute for New Generation Computer Technology

[1,2]4-11-22,Shibaura,Minato-ku,Tokyo 108,Japan
[3]1-4-28,Mita,Minato-ku,Tokyo 108,Japan

We propose the methods of verifying and analyzing program written in a parallel logic language, GHC. This paper describes two methods of support for GHC programming, which detects bugs before executing the program and analyzes of information after execution. One verifies program and calculates the parallelism degree of program by using Petri Nets. The other calulates program-load and indicates it graphically by using program trace information.

1 はじめに

並列プログラミングの特性として,処理が並列に進行し,互いに干渉し合い,また実行順序に再現性がないといった点が挙げられる.このような並列プログラムの設計では,デッドロック等の並列特有のバグやプロセス・スケジューリングに関する問題を含み,逐次プログラムの設計以上に難しいことが指摘されている.そこで,前者の問題に対しては,プログラムを形式的な枠組みでモデル化し,検証する技術が非常に重要であると思われる.また,後者の問題に対しては,自動スケジューリングが非常に困難な問題であるために,プログラムの性能評価をフィードバックすることが有用であると考える.

本稿では,ペトリネット[1]及び並列論理型言語GHC[2]について簡単に説明した後,ペトリネットにおけるGHCプログラムのモデル化及び検証法について述べる.さらに,メタインタプリタを用いて収集したトレース情報をグラフィカルに表示することにより負荷分散設定支援を行う.

2 ペトリネットとGHC

非同期並行系システムをモデル化し解析する手法の一つであるペトリネットと,並列実行を基本とした論理プログラミング言語であるGHCについて簡単に解説する.

2.1 ペトリネット

ペトリネットは,プレース,トランジション,接続行列,マーキングによって表される.ペトリネットの実行はトランジションを発火させることであり,トランジションの発火はその入力プレースからトークンを取り去り,新しいトークンを生成し,出力プレースに分配することである.また,トークンとはプレース内に存在しトランジションの実行を制御するもので,このトークンをプレースに割り当てることをマーキングという.ペトリネットグラフでは,プレースを“○”,トランジションを“|”,プレースとトランジションを結ぶアークを矢印として表記する.ここで,トランジションからプレースへのアークの多重度を示す接続行列,及びその逆向きの行列をそれぞれA^+, A^-とおくと接続行列Aは次式で示される.

$$A = A^+ - A^- \tag{1}$$

マーキングMにおいて,トランジションkが発火するための条件を次式に示す.ここでγ_kは,k番目の要素のみが1でそれ以外がすべて0であるベクトルである.

$$M \geq {}^tA^- \cdot \gamma_k \tag{2}$$

各トランジションの発火回数を集計したベクトルを発火回数ベクトルという.初期マーキングをm_i,目的マーキングをm_j,その間の発火回数ベクトルをγとすると次式が成立する.

$$m_j = m_i + {}^tA \cdot \gamma \tag{3}$$

2.2 GHC

GHCは,並列論理型言語である.以下に,構文と手続き的意味に分けてGHCを簡単に説明する.

[GHCの構文]GHCプログラムは,次に示すガード付き節の有限集合として表される.

$$\underbrace{\underbrace{\underbrace{H}_{head} : - \underbrace{G_1, \cdots, G_m}_{guard\ goals}}_{guard} \underbrace{|}_{commit} \underbrace{B_1, \cdots, B_n.}_{body\ goals}}_{clause}$$

$$m, n \geq 0 \tag{4}$$

ここで,H, G_i, B_jはそれぞれヘッド,ガードゴール,ボディゴールと呼ばれる.また,オペレータ“|”はコミット演算子と呼ばれ,これに先立つ部分をガード部,これに続く部分をボディ部と呼ぶ.

[GHCの手続き的意味]GHCでは,実行に与えられたゴールに対して融合に使えそうな複数の節と並列に融合を試みる.この時,選択された節のヘッドとガードゴールは,その呼出側のゴールを具体化することなく同一化しなければならな

い. もし, この同一化によって呼出側のゴールを具体化しようとした時は, 他の同一化によってゴールが具体化されるまで, その同一化は中断させられる. 次に, ガードが成功した節から一つの節を選択 (コミット) し, ボディゴールと融合に使えそうな節を再び並列に試みる. この時ボディゴールが複数ある場合には並列に実行される. ただし, その節の呼出側を具体化することができるのは, ただ一つの選択された節に限る.

3 ペトリネットを利用したプログラム解析

GHC で記述されたプログラムの検証を目的として, 非同期並行系システムをモデル化し解析する手法のひとつであるペトリネットを用いた. このペトリネットは, 論理と制御からなる一般のプログラムにおける制御部分をモデル化し, 正当性の検証を行おうとする研究に利用されている. これは一階述語論理における導出のモデル化にも利用されており, 論理型言語のプログラム検証にも有効である.

ここでは, GHC プログラムの正当性を可達方程式を用いて調べると共に, 対象としたプログラムにおける並列度を算出する.

3.1 ペトリネットによるモデル化

GHC プログラムをペトリネットに変換するために用いた変換規則と, その解析方法について以下のように検討した.

ある種の GHC のプログラムは, 次のような対応によりペトリネットでモデル化できる.

節 $\Longleftrightarrow$ トランジション
ヘッド, ガード・ゴール $\Longleftrightarrow$ 入力プレース
ボディ・ゴール $\Longleftrightarrow$ 出力プレース

簡単な GHC プログラムをペトリネットでモデル化した例を図 1 に示す.

```
?- p.                          (c0)
p :- true | q(X), r(X).        (c1)
q(X) :- X=a | t.               (c2)
r(X) :- true | X=a, s.         (c3)
r(X) :- true | s.              (c4)
s :- true | t.                 (c5)
t :- true | true.              (c6)
```

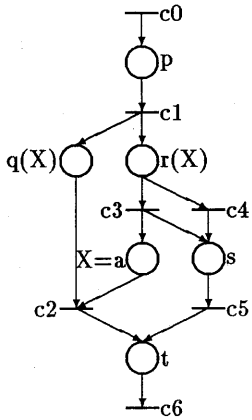


図 1: モデル化の例

3.2 可達方程式に基づくプログラム検証

ここでは, GHC プログラムをモデル化したペトリネットを可達方程式を利用して検証する方法について述べる.

GHC のプログラムに含まれるバグは 2 つに分類される. 一つは, 基本的に実行前にチェックできる, シンタックス等の静的バグで, 比較的用意に検出が可能である. もう一つは, 基本的に実行時までエラーが判明しないものであり, これを動的バグと呼ぶ. 特に, 並列実行を行うプログラムでは, 実行順序が定まらずプログラムを動作させても検出が困難なバグが存在する. そこで, GHC プログラムの制御をペトリネットでモデル化することで, あるカレント・ゴールから他のカレント・ゴールへのリダクション系列が存在するための必要条件を得, 実際にプログラムを動作させることなく, デッドロック等のバグを検出する.

図 1 に示したプログラム例に対して, 式 (3) を用いて考える. プログラムの実行が成功する必

要条件は, $m_i = m_j = 0$ かつゴール節の発火回数を1として, 可達方程式が非負整数解 γ をもつことである. 従って, γ を求めることにより次のことがわかる.

- 解がなければプログラムの実行は失敗する.
- すべての解を通じて発火回数が0である節の実行はデッドロックを引き起こす.

図1に示したペトリネットの接続行列を表1に示す.

表 1: 図1のプログラムの接続行列 A

	p	q(X)	r(X)	X=a	s	t
c0	1	0	0	0	0	0
c1	−1	1	1	0	0	0
c2	0	−1	0	−1	0	1
c3	0	0	−1	1	1	0
c4	0	0	−1	0	1	0
c5	0	0	0	0	−1	1
c6	0	0	0	0	0	−1

表1より, 唯一の解

$$\gamma = {}^t(\ 1\ \ 1\ \ 1\ \ 1\ \ 0\ \ 1\ \ 2\ )$$

が得られる. 実際に, c4 が実行されるとゴール q(X) がデッドロックしてしまう.

3.3 並列度の算出

GHC プログラムを効率的に動作させるためには, スケジューリングを考慮したプログラムの記述が必要である. しかし, GHC プログラムの動作は非決定的であるため, プログラマがプログラムの実行順序を把握した上で, 並列実行の効率を考えたプログラミングを行うことは非常に困難である. そこで, ここでは GHC プログラムをモデル化したペトリネットを用いて, プログラムの並列性を抽出し, 理想的に並列実行された場合の並列度を示すことにより, 並列プログラムのプログラム設計を支援することを考えた.

ここで並列性を抽出するために, 節のリダクション過程を調べることによって, 各時点において並列に実行できる節の数を求める. つまり, ペトリネットに変換した後で, 節の実行をトランジションの発火と捉えて, 発火可能なトランジションの数をその時点における並列度と考えた.

ここで, トランジションが最大に同時発火した場合の発火系列を考えると, 表2に示したようになる.

表 2: 図1のペトリネットの発火履歴

	発火前のトークン数	発火トランジション
1	(000000)	c0
2	(100000)	c1
3	(011000)	c3
4	(010110)	c2, c5
5	(000002)	c6, c6

節の実行は, トランジションの発火に対応しているので, 可達方程式の解の各要素の総和から, プログラムを逐次実行した場合のステップ数 Ns が得られる. また, 同時発火を最大限に許した場合の発火履歴から, 理想的に並列実行された場合のステップ数 Np が得られる. 従って, プログラムの並列度は Ns ÷ Np として求めることができる.

図1のプログラムに関しては, 可達方程式の解から Ns=7, 表2の発火履歴から Np=5 が得られるので, その並列度は 1.4 となる.

4 トレース情報を利用した解析

GHC プログラムをメタインタプリタ上で実行収集したトレース情報を利用して, 各プロセッサにおける負荷量を求め, 負荷の分散状況や推移をグラフィカルに提示することにより, プログラマによる負荷分散設定の支援を考えた. ここでは, 実際にクイーン問題を利用して, 異なる負荷分散指定を行った場合の動作状況から, トレース情報を利用した最適負荷分散の指定方法について検討する.

4.1 負荷量の算出

GHC で記述したプログラムをメタインタプリタ上で実行することにより, プロセッサ毎のゴール・リダクションの履歴や, カレントなゴール・キューの内容等の詳細なトレース情報が得られ

る. この時, ゴール・キューの長さを負荷と考えることによって, 通信時間を 0 とし, 全プロセッサにおけるリダクション時間を等しいとした時の, 理想マシン上における負荷分散の推移を求めることができる.

4.2 各種グラフ表示

対象プログラムの実行により収集した負荷量を時間的推移に従って視覚的に表現することにより, プログラムの負荷分散状況の把握を容易にした. この動作情報は全プロセッサの負荷状況を同時に表示するために, プロセッサ - 時間 - 負荷の 3 次元グラフで示した. この 3 次元グラフを図 2 に示した. ここで, 3 次元グラフ表示に示されている MAX は, 最大ゴール数を示し, TOTAL は, 実行終了までのリダクション回数を示す.

さらに, 任意の時間あるいはプロセッサに注目した表示を行うために, 特定プロセッサに関する負荷推移及び, 特定時間における負荷分散を 2 次元グラフで表示した. 2 次元グラフの表示例を図 2 に示す.

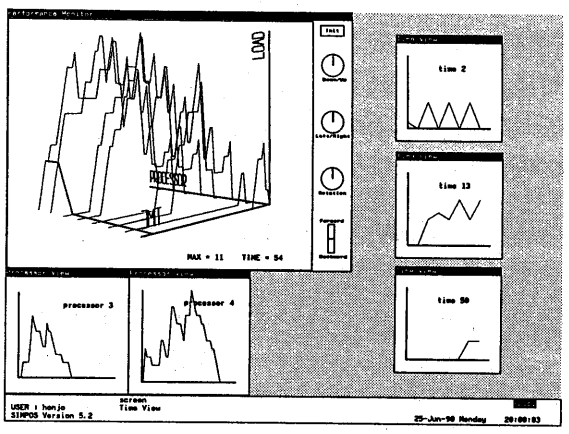


図 2: 各種グラフ表示例

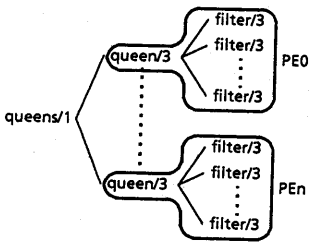
4.3 グラフからトレース情報の参照

負荷情報を表すグラフ上の任意の点を選択することにより, 特定プロセッサ, 特定時間におけるリダクション, ゴール・キューの内容等を表示し参照することができる. この機能により, 負荷情報が示す実行状況の中で, プログラマの意図した負荷分散や静的なスケジューリングが行われなかった部分を詳細に調べることが可能である.

4.4 n クイーン・プログラムにおける負荷分散指定

負荷分散指定を行った n クイーン・プログラムを実行することにより, その負荷分散状況のグラフ表示を用いて, 適正な負荷分散指定を決める. ここで利用した n クイーン・プログラムは, トップレベルの述語 queens より, 述語 queen が n 個, この queen 述語から filter 述語が n 個生成される. 実際に 4 クイーン・プログラムに対して, queen を分散した時と filter を分散した時の 2 種類の負荷分散指定を行った場合の動作状況の 3 次元グラフ表示を図 3, 図 4 に示す.

queen を分散した場合に比べて, filter を分散した例の方が各プロセッサに均等に負荷分散されていることがわかる.



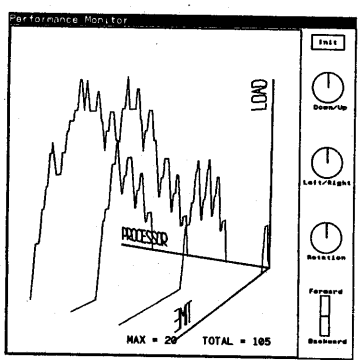


図 3: queen を分散した場合

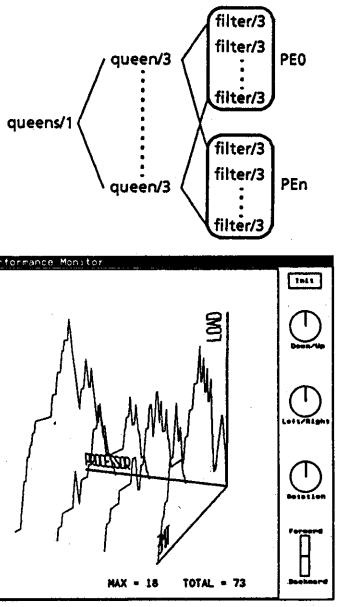


図 4: filter を分散した場合

また, 1 リダクション当たりの最大ゴール数および, 実行終了までのリダクション回数のいずれにおいても, filter を分散する方が少なくなっていた.

次に, 上で示した 2 種類の分散例に加えて, queen 及び filter の両方を分散させた例を Multi-PSI 実機上において実行した. この時の実行時間と, メタインタプリタにおける実行効率を比較したものを表 3 に示す.

表 3: 4queen における実行時間

分散パターン	メタインタプリタ (reduction step 数)	Multi-PSI (msec)
queen のみ	105(100%)	1461(100%)
filter のみ	73(70%)	1100(75%)
queen, filter 両方	54(51%)	1024(70%)

メタインタプリタにおける実行効率の変化は, 実機に相関した評価結果となっていることがわかる. しかし, filter を分散させた場合及び, queen と filter の両方を負荷分散させた場合のいずれにおいても, 実機における実行時間の向上率を上回る値である. これは, トレース情報から算出した負荷が, 通信を考慮に入れない, 理想マシンにおける負荷情報のためである.

5 おわりに

GHC プログラムの設計支援として, ペトリネットを用いたプログラム検証法と並列度の算出法, 及び, トレース情報を利用した負荷の算出法とグラフィカルな提示法を示した. 現在のところ、ペトリネットを用いたプログラム検証において、ペトリネットに変換することのできる GHC プログラムには多くの制限がある。特に、GHC で頻繁に使用されるストリームのペトリネット表現法は今後の課題である。また、トレース情報を利用した負荷分散設定支援においては、クイーン・プログラムを用いた実行時間が示していたように、理想マシンと実マシンにおける実行効率の相違が問題となった。実マシンにおける実行では、プロッセッサ間の通信やゴール毎の重さの違い等から、理想マシンとの違いがあることは明らかである。その中でも通信量が実行時間に与える割合は大きく、通信量を考慮に入れた負荷分散設定を行う必要性が示された。これらの課題については、今後も検討を続けていく。尚, 本研究は第 5 世代コンピュータプロジェクトの一環として行っているものである.

参考文献

[1] Peterson, J. L. : *Petri Net Theory and The Modeling of Systems, Prentice-Hall, 1981*

[2] 古川, 溝口 : 並列論理型言語 *GHC* とその応用, 共立出版, *1987*

[3] 長谷川, 田口, 中島 : ペトリネットを利用した並列プログラムの解析, 第 *3* 回 回路とシステム軽井沢ワークショップ, *pp.199-206(1990)*

[4] 田口, 本城, 中島 : メタインタプリタを用いたパフォーマンスモニタ, *ICOT KL1 Programming Workshop,pp.72-77(1990)*

[5] 田中, 的場 : 変数管理をする *GHC* の自己記述, *ICOT Technical Report, TR-374, 1988*