GHC プログラムの検証と性能評価

吉谷努[1]　田口毅[2]　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

1　はじめに

並列プログラムの設計は，デッドロック等の並列特有のバグやプロセスのスケジューリングの問題のために，逐次プログラム以上に難しいことが指摘されている．前者に対しては，プログラムを形式的な枠組でモデル化し，検証する技術が非常に重要であろう．後者に対しては，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，プログラムの性能を評価し，プログラマに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，並列論理型言語 GHC で書かれたプログラムに対して，検証および性能評価を行なう手法を提案する．なお，紙幅の都合により，GHC とペトリネットの諸概念の定義は文献 [1], [2] に譲る．

2　ペトリネットを用いたプログラム解析

ペトリネットは，非同期並行系をモデル化・解析する手法として考えられたものである．一般に，プログラムは論理と制御からなるが，GHC プログラムの制御部分をペトリネットでモデル化し，種々の検証，並列度の算出を行なう．

2.1　ペトリネットによるモデル化

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

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

簡単な GHC プログラムと対応するペトリネットの例を図 1 に示す．

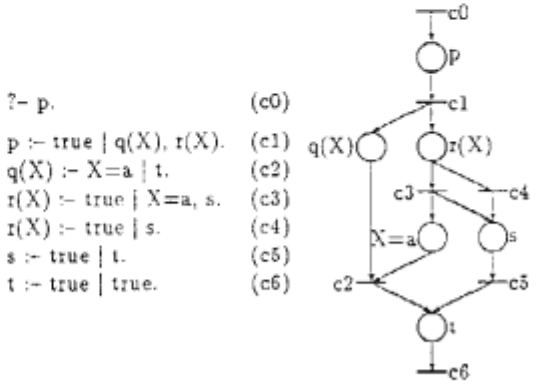

図 1: モデル化の例

2.2　可達方程式に基づくプログラム検証

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が得られる．このため，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ことができる．

定義: 可達方程式　A を接続行列，m_i, m_j を各プレースのトークン数ベクトルとする．m_i 状態から m_j 状態へ遷移可能ならば，

$$m_j = m_i + {}^t A \cdot \gamma$$

w が非負整数解 γ をもつ．ここで，γ は各トランジションの発火回数ベクトルとなる．　□

上の定義より，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分かる．

- 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 のプログラムの接続行列 A は，

Program Verification and Performance Estimation for GHC
Tsutomu YOSHITANI[1]　Takeshi TAGUCHI[2]
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

	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

であるから，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．

2.3 並列度の算出

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 N_s 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 N_p が得られる．従って，プログラムの並列度は $N_s \div N_p$ 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 $N_s = 7$，表 1 の発火履歴から $N_p = 5$ が得られるので，その並列度は 1.4 となる．

表 1: 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

3 トレース情報を利用した解析

プログラムを実行して得られるトレース情報に基づき，各プロセッサの負荷を算出する．さらに，負荷の分散・推移をグラフィカルに提示し，プログラマによる静的なスケジューリング指定を支援する．

3.1 負荷の算出

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，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．このとき，ゴール・キューの長さを負荷と考えることによって，1 リダクションを単位時間とする各プロセッサの負荷推移を求めることができる．

3.2 各種グラフ表示

得られた負荷情報は，そのまま，プロセッサ - 時間 - 負荷の 3 次元グラフで表示できる (図 2)．また，特定プロセッサに関する負荷推移，特定時間における負荷分散は，2 次元グラフで表示する (図 3)．

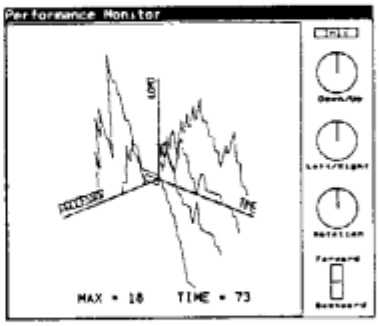

図 2: プロセッサ - 時間 - 負荷の 3 次元グラフ

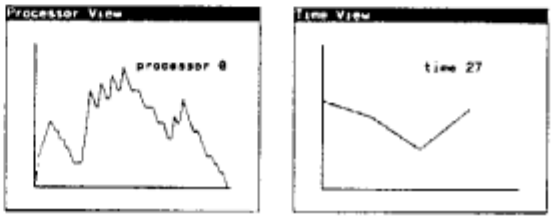

図 3: 負荷推移 (左) と負荷分散 (右) の 2 次元グラフ

3.3 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ある点をポインティングすることにより，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する．この機能により，静的なスケジューリング指定に役立つ情報をプログラマに提示することができる．

4 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，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．なお，本研究は，第五世代コンピュータ・プロジェクトの一環として行なわれたものである．

参考文献

[1] Ueda, K.: Guarded Horn Clauses, TM-103, ICOT, 1985.

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