プログラミング—言語・基礎・実践— 3－12
（1991. 7. 17）

KL1プログラム動作解析ツール パフォーマンスモニタの試作開発

葭谷 努* 中島俊介* 長谷川晴朗** 長谷川隆三***
* 沖通信システム(株)
** 沖電気工業(株)
*** (財)新世代コンピュータ技術開発機構

我々は，並列プログラム設計支援システムの試作開発を行なっており，並列論理型言語KL1プログラムの設計支援システムである，パフォーマンスモニタについて報告する．パフォーマンスモニタは，KL1で書かれたプログラムの，実行時のプロセッサ毎の負荷・プロセッサ間の通信量を，画面にグラフ表示するツールである．実行時のプログラムの動作情報の収集のため，元のプログラムに対し，実行時にリダクションの情報が得られる変換を施し，変換したプログラムを実行させて負荷・通信量の算出を行なう版の試作を，表示機能の改良と合わせて報告する．

KL1 Program Analyzing Tool : Performance Monitor

Tsutomu Yoshitani* Shunsuke Nakajima* Haruo Hasegawa** Ryuzou Hasegawa***
* Oki Telecommunication Systems Co.,Ltd.
** Oki Electric Industry Co.,Ltd.
*** Institute for New Generation Computer Technology

This paper describes a KL1 programming support system "Performance Monitor" which we are developing as a parallel programming support system. The Performance Monitor displays load of each processor and communications between processors graphically, when a program described in a parallel logic language KL1 is executed. The new version of the Performance Monitor collects program execution information by using program conversion, and displaying module is improved.

1 はじめに

KL1 は並列論理型言語 FGHC(Flat Guarded Horn Clauses)[1] を拡張したプログラミング言語である. パフォーマンスモニタは, KL1 で書かれたプログラムの実行時の動作情報を収集して, プロセッサ毎の負荷推移及びプロセッサ間の通信量を算出し画面にグラフ表示することによって, KL1 プログラムのプラグマ指定の効果を確認する際に役立つ, 並列プログラム設計の支援を行なうツールである.

KL1 プログラムの実行時のプロセッサの負荷推移・プロセッサ間の通信量を求めるには, プログラム実行時のゴール起動及び変数具体化等の情報を求めて, プログラムの実行過程を求めることにより, 算出が可能になる. そこで, KL1 プログラム実行時のそれらの動作情報の収集には, 元のプログラムに対して, 実行時にリダクションの情報が得られる様に変換を施したプログラムを, 実際に並列推論マシン・マルチ PSI 上で実行させて, 行なう方法を本システムでは用いる.

パフォーマンスモニタは, 情報収集部, 負荷通信量算出部, 及び表示部より構成され, 本論文の第 2 章～ 4 章はその各部に対応している. 第 2 章ではリダクション情報収集のためのプログラム変換について説明を行なう. また, 第 3 章でリダクション情報からの負荷・通信量の算出を説明する. 第 4 章で算出した負荷・通信量の画面へのグラフ表示を示す. 第 5 章はまとめである.

2 プログラム変換による情報収集

本システムのプログラム変換は, プログラムのリダクションの情報を, 実行時に引数へストリームの形で報告するものである. 本プログラム変換では, リダクション情報の収集を簡単に行なうために, 情報収集のための変換の前に, あらかじめ FGHC プログラムの正規化変換 [2] を行なっておく. 以下, FGHC プログラムの正規化変換について簡単に説明した後, プログラム実行時のリダクション情報を得るための, KL1 プログラムの変換について示す.

2.1 FGHC プログラムの正規化変換

FGHC プログラムの正規形とは, 次の形式の節のことである.

$$H :- G \mid U \cup B .$$

H, G 及び B は単一化を含まないアトムの集合であり, U は単一化の集合,

$$v_1 = t_1, \ldots, v_n = t_n \quad (n \geq 0)$$

である. ここで,

- $v_1, \ldots, v_n$ は異なる変数である
- 各 v_i は H 中に現れ, $t_1, \ldots, t_n$ や B 中に現れない
- ある t_i が変数ならば, t_i は H 中に現れる

を、満たす.

FGHC プログラムの正規化変換は, プログラム節を上述した正規形に変換するものである.

本論文中では, 上述した H,G,U,B の元をそれぞれ, ヘッド, ガード・ゴール, 出力単一化, ボディ・ゴールと呼ぶ.

2.2 リダクション情報収集のためのプログラム変換

ゴール起動及び変数具体化の情報等の FGHC プログラムのリダクションの情報及び, プログラム節の実行したプロセッサ番号を, 実行時に求めることが出来るプログラムに変換する方式を示す. 以下に, ゴール起動の情報を求めるための, 削除・追加されるゴールをリダクション毎にレポートする変換と, 変数の具体化の情報を求めるための, 参照・設定される変数をリダクション毎にレポートする変換をそれぞれ示し, それぞれの変換を同時に行なうレポートの形式について示す. その後, リダクション時の実行プロセッサ番号のレポートについて示す.

2.2.1 削除・追加ゴールのレポート

リダクション毎に削除・追加されるゴールをストリームの形でレポートする様に変換し, レポート・ストリームに対応する引数をヘッドとボディ・ゴールの各々に加え, 「レポート・メッセージ」の出力を行なう単一化を加えるプログラム変換を行なう. このレポートには, プログラムの実行開始から終了までに生成されるゴールの全てを, それぞれ区別して認識する必要がある. そこで, ヘッド及びボディ・ゴールの各々へ, そのゴール毎に固有な ID — ゴール ID と呼ぶ — の付与を行ない, 実行時に生成されるゴールの各々をゴール ID を用いて区別するものとする.

具体的には, 1. ～ 2. に説明する変換をプログラムに対して行なう.

1. ID 付きアトムへの置き換え

プログラムの実行時に生成されるゴールの各々を区別するため, ゴール毎に固有なゴール ID を付与する. そのために, 変換前のプログラムのゴールであるアトム

$$p(t_1, \ldots, t_n)$$

を, ID 付きアトム

$$p(v, t_1, \ldots, t_n)$$

に置き換える. 但し, $t_1, \ldots, t_n$ は変換前のゴールの引数, v は未定義変数とする. KL1 処理系では実行中に生成される変数は, それぞれユニークなものとなるので, この未定義変数をゴール ID として用いることが出来る. この ID 付きアトムへの置き換えは, ヘッドとボディ・ゴールの全てに行なう.

2. レポート・メッセージの単一化の追加

正規化されたプログラム節のリダクションが行なわれた場合に, カレント・ゴール列から削除されるゴールの集合はヘッドであり, カレント・ゴール列に追加されるゴールの集合はボディ・ゴールの集合である. この, リダクション毎の削除・追加されるゴールの情報のレポートを行なうため, 次に示す形式のレポート・メッセージを用いる.

g $(d, [a_1, \ldots, a_l])$

d : 削除されるゴールのゴール ID

$a_1, \ldots, a_l$: 追加されるゴールのゴール ID

レポート・メッセージは, 差分リストを用いたレポート・ストリームで表される. そこで, レポート・ストリームに対応する引数を, ヘッドとボディ・ゴールの各々に加え, レポート・メッセージとボディ・ゴールに加えられた引数との差分リストと, ヘッドに加えられた引数との単一化をプログラム変換時に加える.

変換例 1. 及び 2. を適用した, プログラム節の変換例を次に示す.

```
filter(N,[I|A],C) :-
  I mod N =\= 0 |
    C=[I|B], filter(N,A,B).
```

⇓

```
filter(G,N,[I|A],C,R1,R3) :-
  I mod N =\= 0 |
    R1=[g(G,[Gf])|R2], C=[I|B],
    filter(Gf,N,A,B,R2,R3).
```

2.2.2 参照・設定変数のレポート

リダクション毎に参照・設定される変数を, ストリームの形でレポートする様に変換し, レポート・ストリームに対応する引数をヘッドとボディ・ゴールの各々に加え, 「レポート・メッセージ」の出力を行なう単一化を加えるプログラム変換を行なう. このレポートには, プログラムの実行開始から終了までに生成される項の全てを, それぞれ区別して認識する必要がある. そこで, ヘッド, 出力単一化及びボディ・ゴールの項の各々へ, その項毎に固有な ID — 項 ID と呼ぶ — の付与を行ない, 実行時に生成される項の各々を項 ID を用いて区別するものとする.

具体的には, 1. ～ 3. に説明する変換をプログラムに対して行なう.

1. ID 付き項への置き換え

プログラムの実行開始から終了迄に生成される変数や, 構成される項の各々を区別するため, 項毎に固有な項 ID を付与する. 項 ID を付与することを目的として, 項を次に示す ID 付き項に変換する.

- 項 t が変数または定数ならば ID 付き項は (v, t)
- 項が $f(t_1, \ldots, t_n)$ ならば
  ID 付き項は $f(\{v, v_1, \ldots, v_n\}, t)$
  但し, 項 t_i の ID 付き項を (v_i, t_i) とする

但し, v は未定義変数とする.

ID 付き項 (v, t) または $(\{v, v_1, \ldots, v_n\}, t)$ に対して v をその項 ID, t をその ID 付き項の値と呼ぶことにする. ヘッドとボディ・ゴール中の全ての項をこの ID 付き項で置き換える.

2. 項 ID の単一化

リダクションの実行により, 親ゴール中の変数や新たに生成された変数は, いくつかの子ゴールに引き継がれていく. 正規化されたプログラム節の, ヘッド, 出力単一化及びボディ・ゴール中のすべての引数が ID 付き項に変換した時, 同じ変数を値とする ID 付き項の ID も子ゴールに引き継がれなければならない. そのため, 同じ ID 付き項の値をとる変数の項 ID 同士の単一化をボディ部中に加える必要がある. この場合, 単一化をボディ部中に加えた後には, プログラムの正規化変換を, 再度行なう必要がある.

3. レポート・メッセージの単一化の追加

正規化されたプログラム節で, ヘッド, 出力単一化及びボディ・ゴール中のすべての引数が ID 付き項になっているもののリダクションが行なわれる場合, 参照される変数の集合は,

$$\{(v,t) \in Term(H) | t \notin Var(H) \text{ または } t \in Var(G)\}$$

であり, 設定される変数の集合は,

$$\{(v,t) \in Term(U \cup B) | t \notin Var(U \cup B) \text{ または } t \in Var(G)\}$$

である．但し，$Term(A)$ は，A 中の ID 付き項の集合，$Var(A)$ は，A 中の変数の集合である．
この，リダクション毎の参照・設定される変数の情報のレポートを行なうため，次に示す形式 v のレポート・メッセージを用いる．

v ($[r_1,\ldots,r_m]$,$[s_1,\ldots,s_n]$)

$r_1,\ldots,r_m$ ：参照される変数の項 ID

$s_1,\ldots,s_n$ ：設定される変数の項 ID

レポート・メッセージは変換後のプログラム中では，差分リストを用いたレポート・ストリームで表される．そこで，レポート・ストリームに対応する引数を，ヘッドとボディ・ゴールの各々に加え，レポート・メッセージの単一化をプログラム変換時に加える．

変換例 1., 2. 及び 3. を適用したプログラム節の変換例を次に示す．

```
filter(N,[I|A],C) :-
  I mod N =\= 0 |
    C=[I|B], filter(N,A,B).
```

⇓

```
filter((TN,N),({Tn1,TI,TA},[I|A]),
            (TC,C),R1,R3) :-
  I mod N =\= 0 |
    R1=[v([Tn1,TI,TN],[Tn2])|R2],
    (TC,C)=({Tn2,TI,TB},[I|B]),
    filter((TN,N),(TA,A),(TB,B),R2,R3).
```

2.2.3 ゴール起動・変数具体化情報のレポート・メッセージ

本ツールで行なうプログラム変換では，上に述べた削除・追加ゴールと参照・設定変数の両方を同時にレポートする変換を行なう．この時，レポート・メッセージとして次の様な形式 r を用いる．

r(d,$[a_1,\ldots,a_l]$,
$[r_1,\ldots,r_m]$,
$[s_1,\ldots,s_n]$)

d ：カレント・ゴール列から削除されるゴールのゴール ID

$a_1,\ldots,a_l$ ：追加されるゴールのゴール ID

$r_1,\ldots,r_m$ ：参照される変数の項 ID

$s_1,\ldots,s_n$ ：設定される変数の項 ID

変換例 適用したプログラム節の変換例を次に示す．

```
filter(N,[I|A],C) :-
  I mod N =\= 0 |
    C=[I|B], filter(N,A,B).
```

⇓

```
filter(G,(TN,N),({Tn1,TI,TA},[I|A]),
            (TC,C),R1,R3) :-
  I mod N =\= 0 |
    R1=[r(G,[Gf],[Tn1,TI,TN],[Tn2])|R2],
    (TC,C)=({Tn2,TI,TB},[I|B]),
    filter(Gf,(TN,N),(TA,A),(TB,B),
                R2,R3).
```

2.2.4 その他の情報のレポート

プロセッサ毎の負荷量の算出やプロセッサ間の通信量の算出等を行なうためには，上で述べてきたレポート・メッセージに，リダクション時の実行プロセッサ番号が必要である．そこで，実行プロセッサ番号や，実行された節に付けられたユニークな番号をレポート・メッセージにレポートすることを同時に行なう．実行プロセッサ番号を求めるには，KL1 では組み込み述語を利用して行なえるので，それを B に加える．次の様な，実行プロセッサ番号が報告されるレポート・メッセージの形式 r を用いる．

r(e,c,d, $[a_1,\ldots,a_l]$,
$[r_1,\ldots,r_m]$,
$[s_1,\ldots,s_n]$)

e ：実行ノード番号

c ：実行された節番号

d ：カレント・ゴール列から削除されるゴールのゴール ID

$a_1,\ldots,a_l$ ：追加されるゴールのゴール ID

$r_1,\ldots,r_m$ ：参照される変数の項 ID

$s_1,\ldots,s_n$ ：設定される変数の項 ID

変換例 適用したプログラム節の変換例を次に示す．

```
filter(N,[I|A],C) :-
  I mod N =\= 0 |
    C=[I|B], filter(N,A,B).
```

⇓

```
filter(G,(TN,N),({Tn1,TI,TA},[I|A]),
            (TC,C),R1,R3) :-
  I mod N =\= 0 |
    current_node(E,Dummy),
    R1=[r(E,8,G,[Gf],[Tn1,TI,TN],
                [Tn2])|R2],
    (TC,C)=({Tn2,TI,TB},[I|B]),
    filter(Gf,(TN,N),(TA,A),(TB,B),
                R2,R3).
```

3 負荷・通信量の算出

第 2 節の変換を行なった KL1 プログラムを実行して得られる，プログラムのリダクション時のゴール起動及び変数の参照・設定の情報であるリダクション・レ

ポートより, KL1 プログラム実行時のプロセッサ毎の負荷推移及びプロセッサ間の通信量推移を算出する.

ここで, 正規化されたプログラム節のリダクションが行なわれる時の処理を考えてみる.

1. H 単一化と G 中のゴールを実行するため, ゴール d 中の変数 $r_1, \ldots, r_m$ の値を具体化を待って参照する.
2. ゴール d をカレントゴール列から削除する.
3. U 中の出力単一化を実行し, d 中の変数 $s_1, \ldots, s_n$ の値を設定する. 出力単一化の実行は中断しないので, リダクション時に実行される.
4. B 中のゴール $a_1, \ldots, a_l$ をカレントゴール列に追加する.

この様にして, プログラム実行時の全リダクションについて, 削除・追加されるゴールと参照・設定される変数が, リダクション・レポートより求められることにより, 各リダクションの実行順序が求まり, プログラムの実行過程を求めることができる.

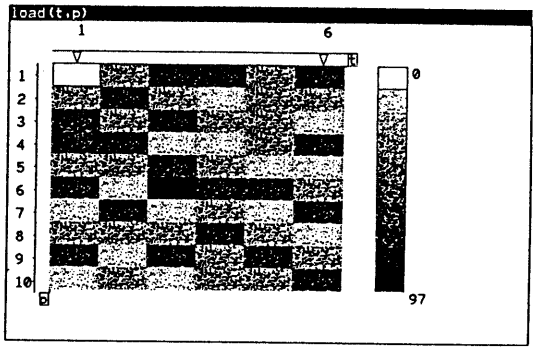
本システムでは, 1 リダクションにかかる時間を一定とみなし, また, プロセッサ間の通信に要する時間を 0 とみなして, 負荷・通信量の算出を行なっている. これは, マルチ PSI 実機での負荷の状態と同じではないが, 言わば, 理想マシン上での負荷・通信量と言う事ができ, プラグマ指定の効果の確認に役立つと考えられる.

3.1 プロセッサ毎の負荷推移の算出

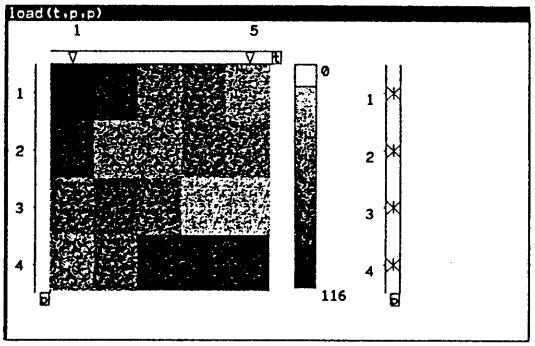
リダクション・レポートから, プログラム実行時の全リダクションについて, 削除・追加されるゴール及び実行プロセッサが分かる. そこで, 実行過程を求めて, 各リダクション毎の追加ゴールの数を求める. 前出の通り, 1 リダクションにかかる時間を一定とみなせば, この各リダクション毎のカレント・ゴールの数が, その実行プロセッサのその時間の負荷量とみなすことができる. この様にして, 全実行過程についてそれぞれの実行プロセッサについて負荷量を算出することにより, プロセッサ毎の負荷推移の算出を行なう.

3.2 プロセッサ間の通信量推移の算出

プロセッサ間の通信は, プロセッサをまたがる, 変数の参照・設定によって起こる. リダクション・レポートから, プログラム実行時の全リダクションについて, 参照・設定される変数が分かる. 上で述べた負荷推移の算出で求めた実行過程を用いて, 各リダクション毎の他プロセッサへ参照・設定している変数の数を求める. 前



(1) プロセスモニタ



(2) コミュニケーションモニタ

図 1: パフォーマンス・モニタの表示例

出の通り，プロセッサ間の通信時間を0とみなせば，この各リダクション毎の他プロセッサへの参照・設定の変数の数が，その実行プロセッサの，その時間での他プロセッサへの通信量とみなすことができる．この様にして，全実行過程についてそれぞれの実行プロセッサについて通信量を算出することにより，プロセッサ間の通信量推移の算出を行なう．

4 画面へのグラフ表示

前節までに述べた方法で算出した，プロセッサ毎の負荷推移及びプロセッサ間の通信量は，2種類のグラフで表示を行なう．各グラフは，複数データを表示し相互に比較することが可能である．また，これらのグラフ上の任意の値や範囲を指定することで，対応するデータを新たにグラフ化することが可能である．
図1.(1)に，プロセッサ毎の負荷推移，図1.(2)に，プロセス間の通信量推移の表示例を示す．この，プロセッサ毎の負荷推移を表すグラフの表示をプロセスモニタ，プロセス間の通信量推移を表すグラフの表示を，コミュニケーションモニタと呼ぶ．

4.1 プロセスモニタ

図1.(1)に示したプロセスモニタは，プロセッサ毎の負荷量を，時間をグラフの横軸，プロセッサ番号を縦軸，負荷量の大小を色の濃淡で表した濃淡グラフである．プロセスモニタでは，マウス操作でデータを選択することにより，以下の1.，2.に示すグラフを表示することが可能である．

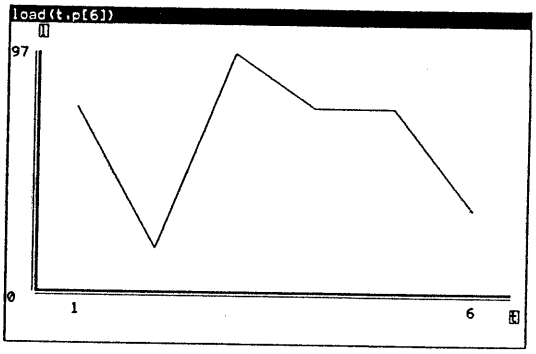
1. プロセッサ負荷推移
   指定したプロセッサにおける負荷推移を，時間を横軸，負荷を縦軸とした折れ線グラフで表示できる．

2. プロセッサ負荷分布
   指定した時間におけるプロセッサ毎の負荷分布を，負荷を横軸，プロセッサ番号をとした棒グラフで表示できる．

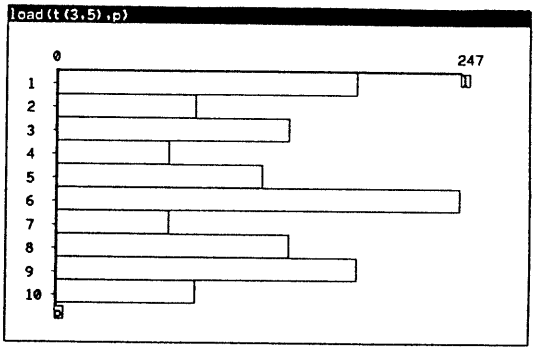
各グラフの表示例を図2.(1)，(2)に示す．

4.2 コミュニケーションモニタ

図1.(2)に示したコミュニケーションモニタは，各プロセッサ間の通信量の推移を時間を横軸，プロセッサの組合せを縦軸に，通信量を濃淡で表した濃淡グラフである．コミュニケーションモニタでは，グラフ上に表示されるメニューを選択することにより，以下の1.～4.に示すグラフを表示することが可能である．

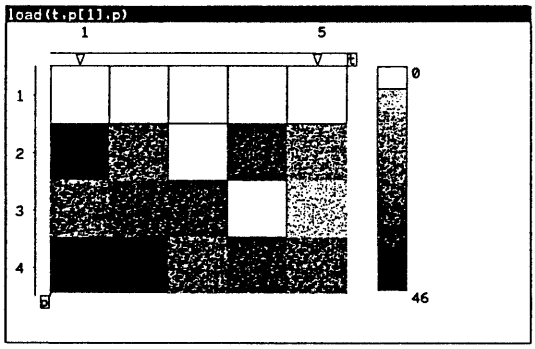


(1) プロセッサ負荷推移

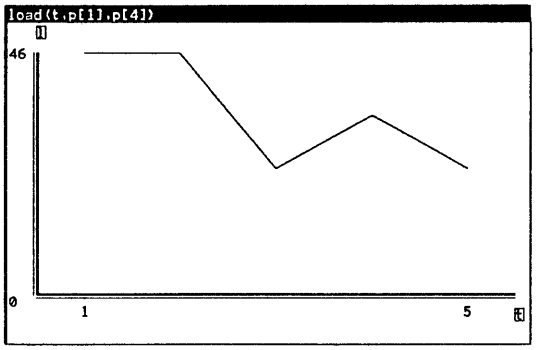


(2) プロセッサ負荷分布

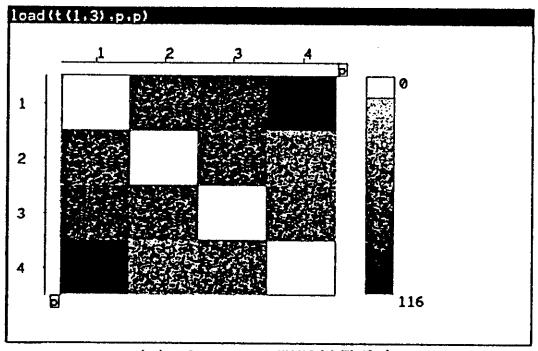
図 2: プロセス・モニタの表示例



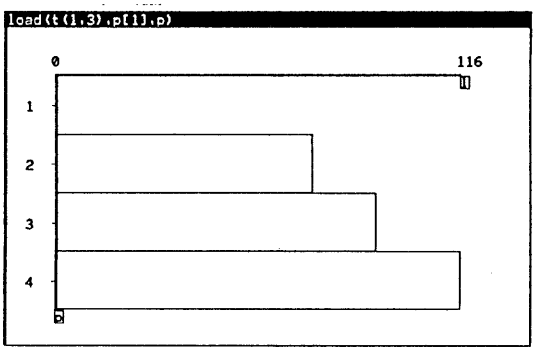
(1) プロセッサ間通信量推移



(2) プロセッサ通信量推移



(3) プロセッサ間通信量分布



(4) プロセッサ通信量

図 3: コミュニケーション・モニタの表示例

1. プロセッサ間通信量推移
   指定したプロセッサの通信量の推移を, 横軸を時間, 縦軸を相手プロセッサ番号とした, 濃淡グラフで表示できる.

2. プロセッサ通信量推移
   指定したプロセッサにおける通信量の推移を, 時間を横軸, 通信量を縦軸にした折れ線グラフで表示できる.

3. プロセッサ間通信量分布
   指定した時間におけるプロセッサ間の通信量の分布を, 横軸に自プロセッサ, 縦軸に相手プロセッサを取り, 通信量を濃淡で表した濃淡グラフで表示できる.

4. プロセッサ通信量
   指定した時間におけるプロセッサ毎の通信量を, 横軸に通信量, 縦軸にプロセッサ番号を取った棒グラフで表示できる.

各グラフの表示例を図 3.(1)–(4) に示す.

5 まとめ

KL1 プログラムの動作情報を変換プログラムの実行によって収集して, プロセッサ毎の負荷推移及びプロセッサ間の通信量推移を算出し, 画面にグラフ表示する並列プログラム設計支援ツール, パフォーマンスモニタについて述べた. 今後の改良点としては, リダクションの重み付けを行なったより実機に近い負荷量の算出方法を検討し, その算出方法を元にした負荷推移の表示をする事, 表示系のユーザインタフェースの更なる向上等を予定している. 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] K.Ueda, Guarded Horn Cluases, Tech. Report TR–103, ICOT, 1985

[2] K.Ueda and K.Furukawa, Transformation Rules for GHC Programs, Proc. of FGCS'88, pp.582–591, ICOT, 1988

[3] S.Nakajima, ペトリネットで表現された GHC プログラムの実行履歴, 情処研報 Vol.90, No.91, 1990