TM-1131

負荷分散設計支援ツール
ParaGraph

相川　聖一、神子　貞弓、久保　秀行、
松澤　史子（富士通）、近山　隆

November, 1991

ICOT
Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan
(03)3456-3191 ~ 5
Telex ICOT J32964

Institute for New Generation Computer Technology

負荷分散設計支援ツール ParaGraph

相川 聖一　神子 真弓　久保 秀行　松澤 史子　†近山 隆
富士通株式会社　†(財) 新世代コンピュータ技術開発機構

概要

疎結合並列計算機上での負荷分散アルゴリズムの設計を支援するために、並列プログラムの動作状況を計測し、視覚的に表示するツール ParaGraph を並列推論マシンのプロトタイプであるマルチ PSI/V2 上に試作した。本ツールでは、プロセッサの稼働状況、プロセッサ間通信の発生状況及び並列プログラムの実行状況を種々のグラフで表示する。本ツールを幾つかの例題に適用した結果、動作状況を種々の観点から視覚的に表示することは、各プロセッサの負荷を均等化する際に非常に有効な手がかりとなり、また、並列処理によって期待される速度向上が得られない場合の要因分析に有効であることが分かった。

Load Distribution Support Tool ParaGraph

Seiichi Aikawa　Mayumi Kamiko　†Takashi Chikayama
Hideyuki Kubo　Fumiko Matsuzawa

Fujitsu Limited
1015, Kamiodanaka Nakahara-ku, Kawasaki 211, JAPAN

†Institute for New Generation Computer Technology
†1-4-28 Mita, Minato-ku, Tokyo 108, JAPAN

To help the programmers design load distribution algorithms on a loosely-coupled multiprocessor, we have developed the system ParaGraph, which visualizes parallel program execution graphically on a prototype, the Multi-PSI/V2. This tool provides graphic displays of the work loads of each processor, inter-processor communication and execution behabior of parallel programs. Experiments with various programs has indicated that graphic displays were helpful to divide work loads into each processor equally and determine which were the bottlenecks in the computation performance on a multiprocessor.

1 はじめに

疎結合並列計算機上で効率良くプログラムを実行するためには、プロセッサ間の通信量を抑えながら計算負荷を各プロセッサに均等に分散することが肝要である。このため、種々の動的負荷分散方式が考案されているが[1]、これらの方式だけで全てのプログラムを効率良く実行するまでには至っていない。このため、プログラマが思考錯誤を繰り返しながら負荷分散アルゴリズムの設計を行なっているのが現状である。そこで、こうした負荷分散戦略の設計を支援するため、プログラム実行時に収集したログ情報に基づいて、動作状況をグラフィカルに表示するツール ParaGraph の試作を行なった。試作は並列推論マシンのプロトタイプであるマルチ PSI/V2[2] 上で行なった。マルチ PSI/V2 は各プロセッサをメッシュ状に結合した疎結合並列計算機で、最大 64 台までのプロセッサを接続することができる。

本稿では、第2章で負荷分散アルゴリズムの設計と疎結合並列計算機マルチ PSI/V2 上での実現方法について述べ、第3章で試作した ParaGraph の機能と動作状況の表示形式について述べる。第4章に本ツールを幾つかの例題に適用した結果を示す。最後に、第5章でまとめと今後の課題について述べる。

2 負荷分散アルゴリズム

2.1 負荷分散アルゴリズムの設計

計算負荷の分散アルゴリズムの設計では、問題 (プログラム) を部分問題 (サブタスク) へ分割する方法と分割された部分問題を各プロセッサへ割り付ける方法について検討する [1]。分割方法の検討では、計算負荷を分散させることによってプロセッサ間通信が増大しないように、ある程度独立に計算可能な単位に問題を分割する必要がある。また、割り付け方法の検討では、ある特定のプロセッサに計算負荷が集中しないように、負荷を均等に各プロセッサに割り付ける必要がある。図 1に分割と割り付けの模式図を示す。図に示すように計算負荷を分割した結果、ある部分問題の粒度が粗く (粗粒度) なる場合は、最も粒度が粗い部分問題を実行するプロセッサの実行時間に全体の計算時間が支配され、全ての部分問題の粒度が細かい (細粒度) 場合には、各プロセッサに部分問題を割り付ける際に生じる通信のオーバーヘッドが増大し、結果的に効率の良いプログラムとはならない。また、適度な粒度の部分問題に分割できた場合でも、割り付け方法に問題がある場合は、ある特定のプロセッサに計算負荷が集中するため効率良くプログラムを実行することができなくなる。従って、負荷分散アルゴリズムの設計では、計算負荷をある程度独立に計算可能な単位に分割し、各プロセッサの負荷を均等化する方法について検討する必要がある。

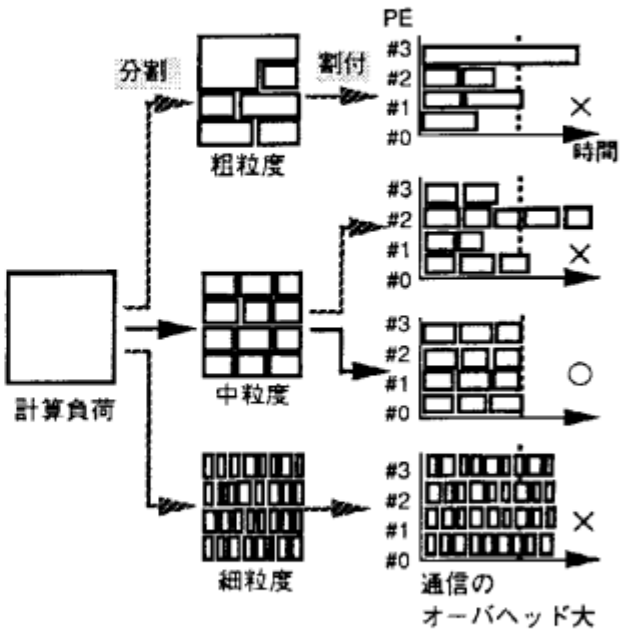


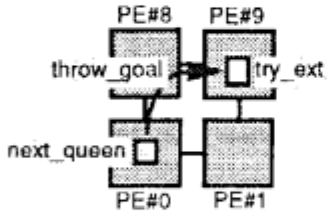
図 1: 計算負荷の分割と割り付け

2.2 負荷分散アルゴリズムの実現方法

マルチ PSI/V2 上で負荷分散アルゴリズムを実現するには、並列論理型言語 KL1[3] でプログラムを作成する。KL1 では、プログラム

```
next_queen(N, I, J, B, R, D, BL):- J>0, D=0 |
        BL={BL0, BL1},          ゴールtry_extを並列に
        R={R0, R1},             実行する。
        BL0=[get(PeNo)|BL2],
        try_ext(N, I, J, B, R0, D, BL2)@node(PeNo),
        next_queen(N, I, "(J-1), B, R1, D, BL1).
```

(a)Nqueen問題のKL1プログラム例



(b)ゴールの分散する様子(PeNo=9)

図 2: KL1 プログラム例

の実行単位であるゴール単位で並列に実行する機能を提供する。図 2 の (a) に KL1 プログラム例を示す [4]。図中の述語 next_queen のボディ部ゴール try_ext にプラグマ "@node(PeNo)" が付加されている。このプラグマが引数 PeNo で指定した番号に対応するプロセッサでゴールを実行させるための負荷分散の指示である。図 2の (b) に述語 next_queen がプロセッサ 0 番でリダクションされた時に、ボディ部ゴール try_next がプロセッサ 9 番 (PeNo=9) へ送信される様子を示す。マルチ PSI/V2 上での KL1 処理系 [5] では、ゴールの分散は対応するプロセッサへゴール送信用メッセージ throw_goal を送出することで実現される。この他の通信用メッセージとして、ガード部ゴールの実行時に他のプロセッサが保持する変数値を参照する外部参照メッセージ read、外部参照メッセージを受け取ったプロセッサで要求があった変数の値を問い合わせ元のプロセッサに返答するメッセージ answer_value、他のプロセッサにユニフィケーションを依頼するメッセージ unify 等がある。従って、負荷分散アルゴリズムを実現するためには、上述のプロセッサ間通信を考慮した上で並列に実行可能なゴールにプラグマを付加し、引数に指定するプロセッサ番号を実行時にプログラムで算出する必要がある。

3 システム概要

ParaGraph では、並列プログラムの性能評価を支援するために、各プロセッサの稼働状況、プロセッサ間通信の発生状況及び並列プログラムの実行状況をマルチ PSI/V2 上で計測し、計測した実行状況をグラフィック表示する機能を提供している。

3.1 実行状況の計測

ParaGraph では、並列プログラム実行時に一定時間間隔毎に以下で述べる情報を計測し、ログ情報を作成する。以下において、各時間間隔に 1 から順に通し番号を割り付けたものをサイクルと呼ぶ。

(a) プロセッサの稼働状況の計測

プロセッサの稼働状況として、各サイクル内における各プロセッサのアイドル時間と非アイドル時間の累計を計測し、稼働率を算出する。非アイドル時間には、プログラムの実行時間、通信用メッセージの送受信の処理時間及びガーベジコレクションの時間が含まれる。稼働率は、"稼働率 [%]" = 非アイドル時間 /(非アイドル時間 + アイドル時間)×100 で算出している。

(b) プロセッサ間通信の発生状況の計測

プロセッサ間通信の発生状況として、各サイクルにおいて各プロセッサで送受信処理を行なった通信用メッセージの送信回数と受信回数を計測する。プロセッサ間で遣り取りされる通信用メッセージの中で、主に負荷分散指示に基づくゴールの送信用メッセージ throw_goal、外部参照メッセージ read、外部参照要求への返答用メッセージ answer_value 及びユニフィケーションの依頼用メッセージ unify を計測する。

(c) プログラムの実行状況の計測

並列プログラムの実行状況の計測では、各サイクルにおいて各プロセッサで実行したゴールのリダクション数とサスペンション数を計測し、述語名別に集計する。従って、同一述語名のゴールは、同じ述語として集計される。

3.2 実行状況のグラフィック表示

実行状況のグラフィック表示では、収集されたログ情報から、何が(What)が、いつ(When)、どのプロセッサ(Where)で、どれくらい処理されたかに着目して、プロセッサの稼働状況、プロセッサ間通信の発生状況及び並列プログラムの実行状況を以下で述べる表示形式でウィンドウに描画する。

(a)What×When の表示形式

本表示形式では、時間の経過に伴う処理量の変化より性能が低下する箇所を判別するために、何(What)が、いつ(When)、どれくらい処理されたかを折れ線グラフで表示する。プロセッサの稼働状況の表示では、各プロセッサの時間の経過に伴う稼働率の変化を表示し、プロセッサ間通信の発生状況の表示では、各プロセッサで送信または受信した通信用メッセージの送信回数と受信回数の変化を表示する。並列プログラムの実行状況の表示では、時間の経過に伴う述語のリダクション数とサスペンション数の変化を表示する。これにより、例えば、並列プログラム実行時に強い同期やプロセッサ間通信による遅延が生じると、それらの要因が解除されるまで述語の実行は中断されため、時間の経過に伴う述語の実行回数の変化を表示することによって同期や遅延の有無を判別することができる。本表示形式には、What 別に個別の折れ線グラフで表示する What×When の表示形式と、全ての What を同一折れ線グラフ中に積み重ねて表示する Overall What×When の表示形式がある。図3に各述語の実行状況を個別の折れ線グラフで表示した What× When の表示例(横軸: サイクル数, 縦軸: リダクション数)を、図4に述語の実行状況を一つの折れ線グラフにまとめて表示し、プログラム全体の実行状況を表す Overall What×When の表示例(横軸: サイクル数, 縦軸: リダクション数)を示す。

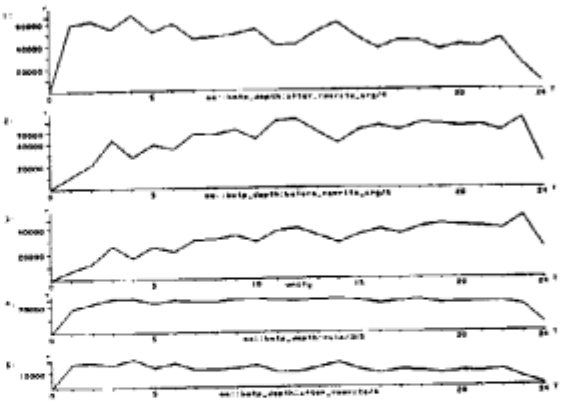


図 3: What×When の表示形式

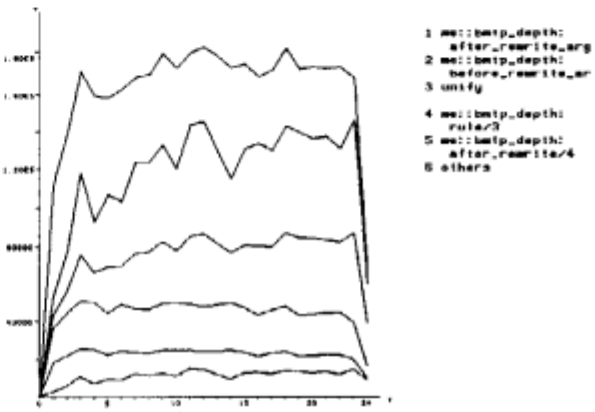


図 4: Overall What×When の表示形式

(b)When×Where の表示形式

本表示形式では、時間の経過に伴う各プロセッサ上での処理量の変化より負荷の分散状況を把握するために、いつ(When)、どこで(Where)、どれくらい処理されたかを濃淡グラフで表示する。プロセッサの稼働状況の表示では、時間の経過に伴う各プロセッサの稼働率の変化を表示する。プロセッサ間通信の発生状況の表示では、各プロセッサで送信または受信した通信用メッセージの送受信回数の変化を表示する。

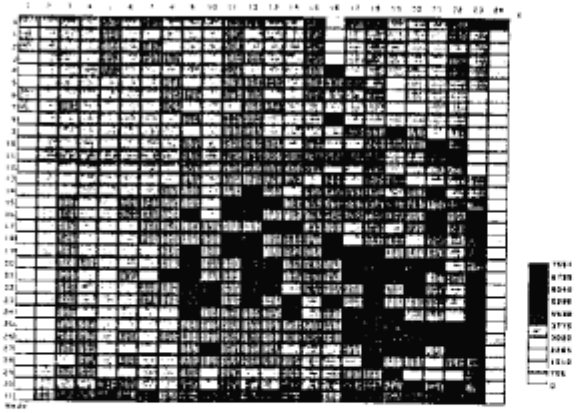


図 5: When×Where の表示形式

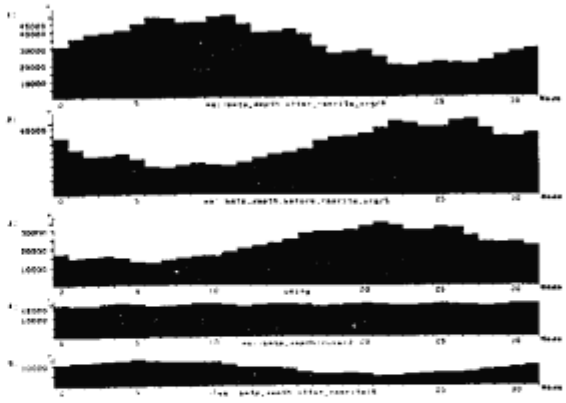


図 6: What×Where の表示形式

並列プログラムの実行状況の表示では、各プロセッサ上での時間の経過に伴う述語のリダクション数とサスペンション数の変化を表示する。これにより、例えば、時間の経過に伴う各プロセッサ上での述語のリダクション数の変化が表示されるため、述語の分散実行状況の良し悪しを判別することができる。濃淡グラフでは、横軸にサイクル数、縦軸にプロセッサ番号を表示し、グラフの右横に色の濃淡と処理量との対応関係を表すメータを表示する。処理量は 10 段階に分割されて、各段階に応じて色が割り当てられる。色が黒いほど処理量が多いことを表す。但し、カラーディスプレイ使用時は濃淡を赤、青、黄等の色の違いで表現する。図 5に並列プログラムの実行状況の When×Where の表示例を示す。図より、全プロセッサがほぼ黒いパターンで表示されていることから、各プロセッサ上でのプログラムが均等に実行されていることが分かる。

(c) What×Where の表示形式

本表示形式では、各プロセッサ上での負荷バランスを判別するために、何 (What) が、どこで (Where)、どれくらい処理されたかを棒グラフ表示する。プロセッサの稼働状況の表示では、全サイクルを通しての各プロセッサの平均稼働率を表示し、プロセッサ間通信の発生状況の表示では、各プロセッサ送信または受信した

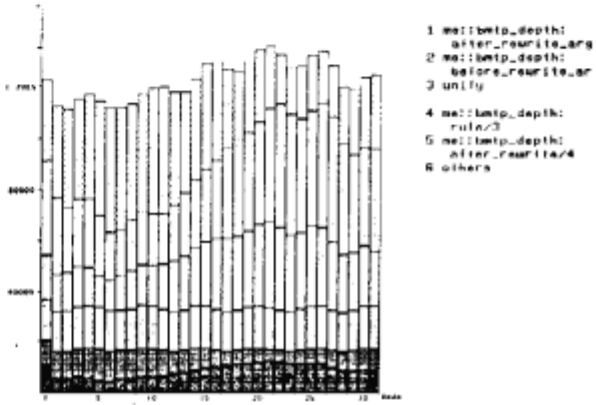


図 7: Overall What×Where の表示形式

通信用メッセージの総送受信回数を表し、並列プログラムの実行状況の表示では、各プロセッサ上での述語の総リダクション数または総サスペンション数を表示する。これにより、例えば、KL1 プログラムの実行結果として、どの述語が、各プロセッサ上で、何回リダクションしたかを参照することにより各プロセッサ上での述語の負荷バランスを把握することができる。本表示形式には、各 What の負荷バランスを個別の棒グラフで表示する What×Where の表示形式と全体の負荷バランスを同一棒グラフ中に積み重ねて表示する Overall What×Where の表示形式がある。Overall What×Where の表示形式では、処理量の多い順に個別のパターンを割り付け、グラフ中の上から順に表示する。図 6に各プロセッサ上での述語の実行状況を示す

What×Where の表示例 (横軸: プロセッサ番号, 縦軸: リダクション数) を示す。また、同じ例題の Overall What×Where の表示例 (横軸: プロセッサ番号, 縦軸: リダクション数) を図7に示す。図より、各述語のリダクション数は各プロセッサ上でばらつきがあるが、プログラム全体では各プロセッサ上で実行された述語の総リダクション数が同程度であることから負荷バランスが良いことが分かる。

3.3 ユーザ・インタフェース

実行状況の計測は、PIMOS[6] が提供するデバッガから計測用コマンドを入力することによって開始し、プログラムの実行終了時に計測処理を終了する。また、無限ループやデッドロック状態に陥ったプログラムの実行状況を表示するために、プログラム実行時に計測処理を強制的に終了し、それまでに計測された情報を収集する機能も提供する。実行状況の表示では、計測したログ情報を表示する表示形式をメニューで選択することによってウィンドウに描画する。また、収集したログ情報の中から特定の What(プロセッサ, メッセージ, 述語) を選んで各表示形式で表示するためのインタフェースを提供し、What 指定の When×Where の表示を可能にしている。

4 適用例

以下において、幾つかの例題の実行状況を ParaGraph で表示した結果を示す。なお、実行状況は1サイクルを2秒単位で収集したものである。

(a) サブタスクの割り付け

図8に論理式の簡約化プログラムの動作例 (When×Where, 横軸: サイクル数, 縦軸: リダクション数) を示す。本プログラムでは、入力論理式を部分論理式に分割し、各部分論理式を並列に簡約化する。図より、各サイクルにおいて

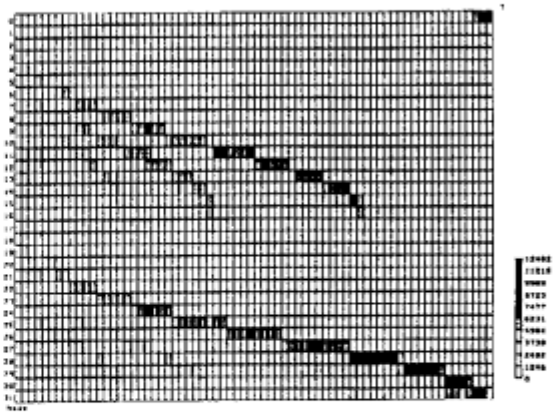


図 8: プログラム全体の実行状況

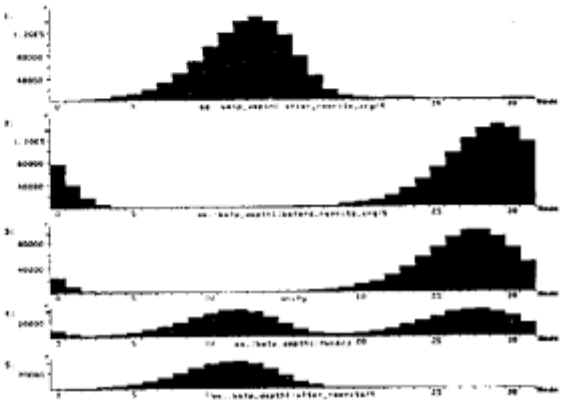


図 9: 各述語の負荷バランス

ある特定のプロセッサに負荷が集中していることが分かる。また、図9に各プロセッサで実行した各述語の負荷バランスを個別に表示 (What×Where, 横軸: プロセッサ番号, 縦軸: リダクション数) すると、述語名から論理式を変換する一連の述語がある特定のプロセッサで実行されていることが分かる。従って、これらの述語を各プロセッサに均等に割り付けるように改良すれば良いことが分かる。

(b) サブタスクの分割

図10に N クィーン問題の動作例 (When×When, 横軸: サイクル数, 縦軸: プロセッサ番号, 処理量: プロセッサの稼働率 [%]) を示す。本プログラムでは、I 行 J 列 ($1 \le I \le N$, $1 \le J \le N$) にクィーンを置いた場合の残りのクィーンの置き

方を求めるサブタスクを生成し、各サブタスクを並列に実行し、全解探索する。サブタスクの生成はプロセッサ0番で行なっている。図よりプロセッサ0番の稼働率が高く、その他のプロセッサはアイドル状態に近いことが分かる。また、プロセッサ間通信の発生状況を図11(What×When, 横軸: サイクル数, 縦軸: 送信回数)に示す。図より、ゴールの送信用メッセージthrow_goalが全サイクルに渡ってプロセッサ0番から送信されていることがわかる。これより、0番以外のプロセッサでサブタスクが実行される速度に比べて、0番でサブタスクが生成される速度が遅いため、他のプロセッサは実行するサブタスクが供給されずアイドル状態になる(稼働率が低下する)サブタスク生成ボトルネックが生じていることが分かる。このため、サブタスクを生成するプロセスを複数のプロセッサに分割する必要があることが分かる。

(c) プロセッサ間通信による遅延

整数列1,2,···,Nを生成し、出力ストリームに送信する生産者プロセス(述語名generator)をプロセッサ0番で実行し、生成された整数列をストリームを介して受け取り、消費する消費者プロセス(述語名consumer)をプロセッサ1番で実行した例を図12(What×When, 横軸: サイクル数, 縦軸: リダクション数)に示す。図より、生産者プロセスは約1サイクルで実行を終了しているが、消費者プロセスは実行が終了するまで7サイクル必要なことが分かる。また、図13にプロセッサ間通信の発生状況の表示例(What×When, 横軸: サイクル数, 縦軸: 受信回数)を示す。図より、全サイクルに渡ってプロセッサ間で外部参照メッセージreadと返答用メッセージanswer_valueが発生していることから、各サイクルで転送可能なデータ量(整数列)の限界から、消費プロセスは必要なデータが到着するまで実行されず、その結果述語consumerのリダクション数が一定回数以上にならないことが分かる。

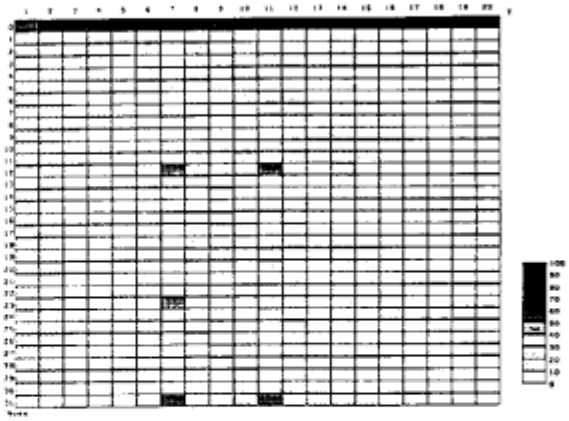


図 10: プロセッサの稼働状況

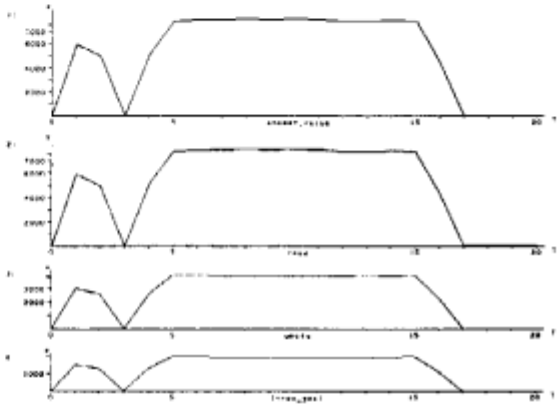


図 11: プロセッサ間通信の発生状況

5 おわりに

疎結合並列計算機マルチPSI/V2上に試作した負荷分散設計支援ツールParaGraphの機能と適用例について述べた。これまでに幾つかの例題に適用した結果、並列プログラムの実行状況を視覚的に表示することは、各プロセッサの負荷を均等化する際に、どの述語のプロセッサへの割り付け方法を改良すべきかが明確になるため、非常に有効なことが分かった。また、一つの表示だけではパフォーマンスのボトルネックが明確にならない場合でも、プロセッサの稼働状況、プロセッサ間通信の発生状況及び述語の実行状況の表示を組み合わせることによって、述語のリダクション数が低下する要因や分割した部分問題の粒度差のために生じる性能低下等の要因の分析が容易になることが分かった。

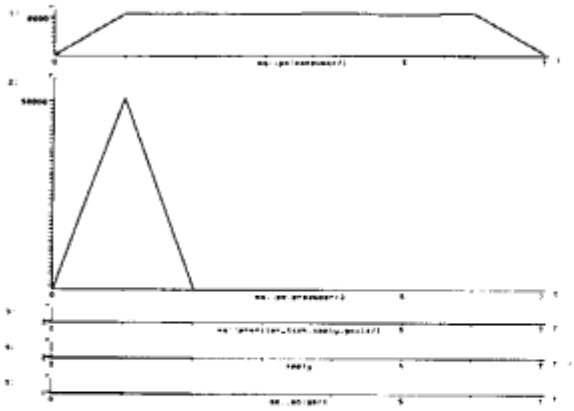


図 12: プログラムの実行状況

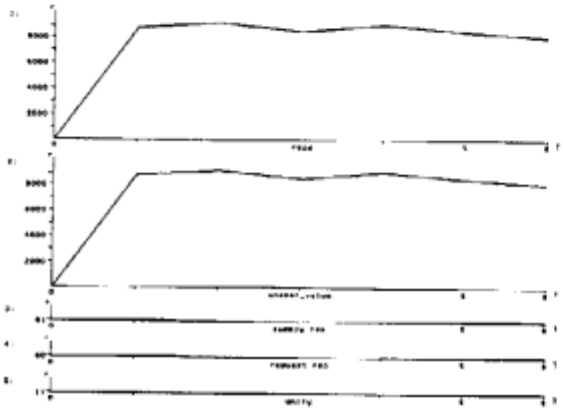


図 13: プロセッサ間通信の発生状況

しかし、計算負荷をある程度独立に計算可能な単位すなわちプロセッサ間通信を引き起こさない単位に分割する際には、現在の表示からはどのゴールがどのデータを参照するためにプロセッサ間通信が発生するのかを明確に表示できないため、分割方法の検討を支援するには今後改良の余地があることが分かった。従って、今後の課題としては、プロセッサ間通信の発生状況をより細かく表示するために、実行状況の計測では、非アイドル時間をさらに細かく分類し、不要なメモリを回収するガーベジコレクションの処理時間、プログラムの実行処理時間、通信用メッセージの送信と受信の処理時間等のように系統立てて実行状況を計測する必要がある。実行状況の表示では、プロセッサ間通信に関する表示の充実を図ると供に並列推論マシン PIM のようにプロセッサ台数が数百台規模に増加した場合でも各プロセッサ上でのプログラムの実行状況が把握できるように現在の各表示形式を改良する必要がある。また、並列プログラム実行時にリアルタイムで表示するツールについても検討する必要がある。本ツールは平成３年１月中旬から PIMOS の一部として第五世代コンピュータプロジェクトに参加している研究者向けにリリースを開始している。今後は、ユーザからの意見に基づいて、より効果的なパフォーマンスデバッキング環境を目指して改良を進めていく予定である。

謝辞

本システムの試作を手伝って頂いた富士通ソーシアルサイエンスラボラトリの角田篤泰氏、また種々の御助言を頂いた応用技術株式会社の中尾浩一氏並びに ICOT 第二研究室の方々に感謝致します。

参考文献

[1] M. Furuichi, et al. , "A Multi-Level Load Balancing Scheme for OR-Parallel Exhaustive Search Program on the Multi-PSI", ICOT TR-526, 1989

[2] K. Nakajima, et al. , "Distributed Implementation of KL1 on the Multi-PSI/V2", In Proceedings of the Sixth International Conference on Logic Programming, 1989

[3] K. Ueda, and T. Chikayama, "Design of the Kernel Language for the Parallel Inference Machine", The Computer Journal, December 1990

[4] ICOT 第四研究室, "KL1 プログラミング 入門編／初級編／中級編", ICOT TM-0722, 1989

[5] K. Nakajima, et al. , "Evaluation of Interprocessor Communication in the KL1 Implementation on the Multi-PSI", ICOT TR-531, 1990

[6] T. Chikayama, et al. , "Overview of the parallel inference machine operating system", In Proceedings of the International Conference of FGCS, 1988