プログラミング-言語・基礎・実践- 8－19
(1992. 8. 21)

高並列プログラムのパフォーマンス・デバッギング・ツール paf

白木 長武　舘村 純一　小池 汎平　田中 英彦
{shiraki,tatemura,koike,tanaka}@mtl.t.u-tokyo.ac.jp

東京大学工学部 電気工学科

並列プログラムのパフォーマンスを決定する要因は複雑である。十分なパフォーマンスが得られない場合にプログラマがその原因を突き止めるためには、パフォーマンス・デバッギング用のツールが必要である。我々は Committed-Choice 型言語 Fleng のプログラムのアルゴリズムの問題点を発見するのに有用と思われるツール paf を設計し実装をおこなった。Paf では仮想時刻という概念を導入し、実行環境の影響を受けずにプログラムの挙動を把握することができる。本稿では paf の詳細とその使用経験について報告する。

Paf: Performance Debugging Tool for Highly Parallel Programs

Osamu SHIRAKI　Jun'ichi TATEMURA　Hanpei KOIKE　Hidehiko TANAKA
{shiraki,tatemura,koike,tanaka}@mtl.t.u-tokyo.ac.jp

Department of Electrical Engineering, The University of Tokyo,
Hongo 7-3-1, Bunkyo, Tokyo, 113 JAPAN

The factors which determine the performance of parallel program are complicated. When a parallel program can't achieve enough performance, the tool for performance debugging is necessary to find the cause out. We designed and implemented a performance debugging tool "paf" for Committed-Choice Language Fleng. "Paf" is useful for discovering problems of program's algorithm. We introduce an concept of virtual-time to observe the action of a program without the influences of executing environment. In this paper, we show the details of "paf" and the examples of using it.

1 はじめに

並列プログラムにおいて十分なパフォーマンスが得られないことがあるが、その理由はさまざまである。通信などのオーバーヘッドの増加や、プロセッサへの不適切な負荷分散がその原因となることもあるが、アルゴリズムそのものに逐次実行部分を多く含んでしまうために並列プログラムの性能が得られないという事態を経験することも多い。

前者の負荷分散などの問題は、適切なものをユーザーが選択するなどの余地はあるものの、基本的にはシステム側の責任といえるが、後者のアルゴリズムの問題は一般のユーザー側の責任である。

ユーザーがプログラムのアルゴリズムの問題を効率的かつ効果的に解決するためには、パフォーマンス・デバッギング用のツールが必要となる。今回、アルゴリズムの問題を発見するのに有用と思われるパフォーマンス・ツールを設計し一部を実装したので、その詳細及び使用経験について報告する。

2 章で、パフォーマンス・デバッギングの問題点について述べる。3 章で、対象としている言語 Fleng について説明する。4 章で、以上で指摘した問題点を解決するパフォーマンス・デバッギング・ツール paf の設計について述べる。5 章で、paf が提供するディスプレイを示す。6 章で、paf の適用例を紹介する。

2 パフォーマンス・デバッギング

パフォーマンス・バグ

並列プログラミングには

- 正しい結果を出す
- 十分な速度を出す

という二つの困難がある。

複数のタスクが協調して動作しなければならないので、プログラマはそれらの間の通信について注意を払わなければならない。しかし、正しい同期がおこなわれて正しい結果が得られたとしても、過度な同期をとってしまったためにプロセッサを有効に利用できなくなり、全体のパフォーマンスを落としてしまうことがある。

マルチ・プロセッサ・システムはシングル・プロセッサの処理能力の限界を越えるために開発された以上、マルチ・プロセッサでは相応のパフォーマンスが得られることが期待され、その要求を満たさないプログラムはパフォーマンス面でのバグがあると言える。

パフォーマンス・バグの原因

並列プログラムのパフォーマンスが出ない原因はさまざまである。

- アルゴリズム
- スケジューリング
- 負荷分散
- データの配置
- コンパイラ
- マシンの形態 (プロセッサ数、処理速度、結合)

一般にこれらの要因は互いに独立ではなく、ある部分の修正は他に影響をおよぼす。

パフォーマンス・デバッギングの手法

パフォーマンス低下の原因を探る手法として次の 2 つがある。

プロファイリング プログラムの部分ごとにパフォーマンス・データを集計する。

トレーシング プログラムの実行に沿って各タスクの挙動を記録し、検証する。

前者のプロファイリングに分類されるツールとしては、KL1 のパフォーマンス・チューニング・ツール ParaGraph[1] などがある。それは、プロセッサの負荷や通信量などのプロファイル・データを測定し、それらを視覚化することができる。

プロファイリングではプロセッサの負荷などの大域的な情報が得られるが、タスク間の依存関係などの細かな情報は得られない。

後者のトレーシングをおこなうツールとしては、MultiLisp のパフォーマンス視覚化ツール ParVis[2],MulTVision[3] などがある。

トレーシングにより、各タスクの挙動を視覚化し、パフォーマンス低下をひきおこす部分を発見することができる。

視覚化

並列プログラムは実行の非決定性などのため、挙動を把握しにくく、デバッグが困難である。また、一般にデバッギング用の情報は膨大になるので、プログラマが有用な情報を得るには、情報のフィルタリングが必要である。したがってプログラマが実行状況を把握するためには適切な視覚化が不可欠である。

パフォーマンス・チューニング

実際のマシン上で実行しその様子を観察するパフォーマンス・モニタを使って、プロセッサの負荷や通信量などを測定し、その解析によりパフォーマンスを向上させる手法がある。

その手法では、実行速度を決定するさまざまな要因の混在した状況での実行結果が観察されるので、パフォーマンスが出ない原因の絞り込みが難しい。また、特定の実行環境でのチューニングであるので、チューニングされたプログラムが、他の実行環境においても良いパフォーマンスが得られるとは限らない。

3 Fleng

本ツールの対象としている言語は Committed-Choice 型言語 (CCL) Fleng[4] である。

Concurrent Prolog や GHC などの CCL は論理型プログラムを並列に実行するためガードの概念を導入して通信・同期が記述できるように制御機能を強化した並列論理型言語である。Fleng はこの CCL の一つであるが、他の CCL に較べてその言語仕様が簡潔な点が特徴である。Fleng はガードゴールを持たず、ヘッドのみがガードの働きをする。よってヘッド・ユニフィケーションだけで定義節がコミットされる。

Fleng の実行イメージを図 1 に示す。

Fleng の実行は、ゴール生成とそのリダクションによって進行し、変数の束縛によって実行が制御される。したがって Fleng には、tree 状のコントロール依存関係と、複雑なデータ依存関係とが存在する。

また、粗粒度あるいは中粒度のプログラムと比較すると、細粒度のプログラムは、生成されるタスクの数が多く、実行単位が小さい。したがってトレース・データを記録する場合に、多くの情報を記録しなければならず、また情報の記録にかかるオーバーヘッドが実行状況に影響を与えてしまう、という問題点が出てくる。

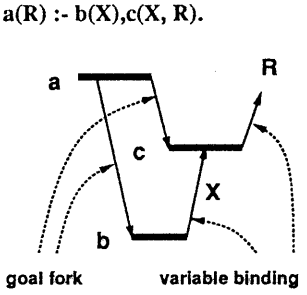

図 1: Fleng の実行イメージ

節 a がコミットされ、ゴール b,c を生成する。コミットされた b は変数 X を束縛し、その結果 c がコミットされる。c は変数 R を束縛する。

4 Paf

コントロールやデータの依存関係によるパフォーマンス低下を発見するには、トレーシングが必要であり、膨大なデータをフィルタリングする適切な視覚化が必要であることは上に述べた。また並列プログラムのパフォーマンスを決定する要因は複雑であり、パフォーマンス・チューニングでは実行環境によっては効果が違ってしまうことも述べた。

したがってパフォーマンス・デバッガへの要求は、

- バグの絞り込みを支援する。
- (可能なかぎり) 実行環境に影響されない。

ということになるであろう。

実行環境の影響を少なくする、並列プログラムのパフォーマンス・デバッギングのスタイルは

1. まずアルゴリズムの問題を解決
2. その後に他の問題を処理

というステップとなる。そのために、アルゴリズムに起因するパフォーマンス・バグを発見できるツールである paf を開発している。

実行環境に影響されずアルゴリズムの問題点を発見するために、paf では以下に述べる実行モデルを用いている。

4.1 Paf における Fleng 実行モデル

プログラマが自分の設計した並列アルゴリズムに内在する問題点を明解に把握するためには、並列アルゴリズムの性能を、プロセッサ台数やスケジューリング戦略などシステム・パラメータと独立に評価できることが望ましい。

そのようなパラメータの制約を受けないモデルは

- プロセッサ数は無限大
- 理想的なプロセス・スケジューリング

である。このモデルでは、リダクション可能なゴールはすぐに実行される。

4.2 仮想時刻

以上のようなモデル上でのプログラムの実行を観察するために、仮想時刻及び仮想並列度という概念を導入する。

仮想時刻とは以上のようなモデル上でプログラムを実行をしたときの時刻であり、仮想時刻から計算される並列度を**仮想並列度**と呼ぶ。

Fleng プログラムの実行において、ゴールリダクションの仮想時刻、プログラム実行の仮想並列度は以下のように計算される。

あるゴールがリダクションされた結果生成されたゴールは、それをヘッドに持つ定義節の引数のユニフィケーションを待つ。ヘッドユニフィケーションに成功した定義節のうち 1 つがコミットされ、ボディ・ゴールのリダクションが開始される。実行ではゴールの生成および変数のユニフィケーションが行なわれる。図 2 にゴールが生成されてから、そのリダクションが終了するまでの様子を示す。

ゴールが生成され (T1)、サスペンション・チェックが行なわれる (T2,T3,T4)。この図ではチェックにかかる時間は 1 と仮定している。チェックの結果コミットされると (CT)、ボディの実行を開始する。実行ではゴールの生成と変数の束縛が行なわれる。

各ゴールは生成された時刻から終了まで時間をカウントし、ボディ実行におけるゴールの生成および変数の束縛について、その時刻を記録する (図 3)。仮想時刻は、実際の処理系で処理された時刻とは無関係になる。

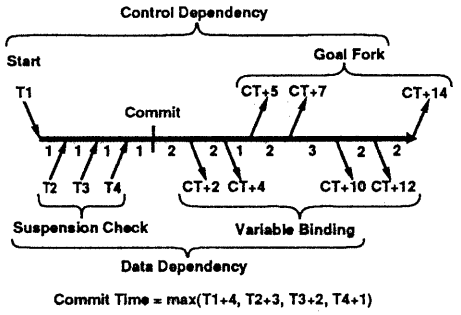

図 2: Fleng のゴール・リダクション

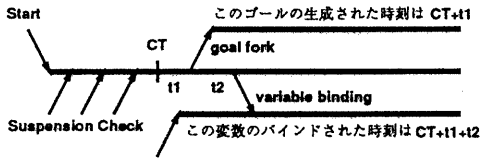

図 3: 仮想時刻の記録

4.3 Pafleng

以上の実行モデルにおけるトレース・データを得るために、仮想時刻を管理し、出力する機能を Fleng インタプリタに追加した。このインタプリタを pafleng と呼んでいる。Pafleng はゴール・リダクションおよび変数の束縛時に、同時にタイム・スタンプを記録し、リダクション終了時にその情報を出力する。

Pafleng は、各ゴールについて、

リダクション ID, 親リダクション ID,
開始時刻, 終了時刻, 述語名,
ヘッド引数の束縛情報
(束縛したリダクション ID および時刻)

を出力する。

5 Paf のディスプレイ

Paf のディスプレイは、メイン・ウィンドウ、ゴール選択ウィンドウ、フロー・ウィンドウ、ヒストグラム・ウィンドウで構成される (図 4)。

以下に、各ウィンドウとその役割について述べる。

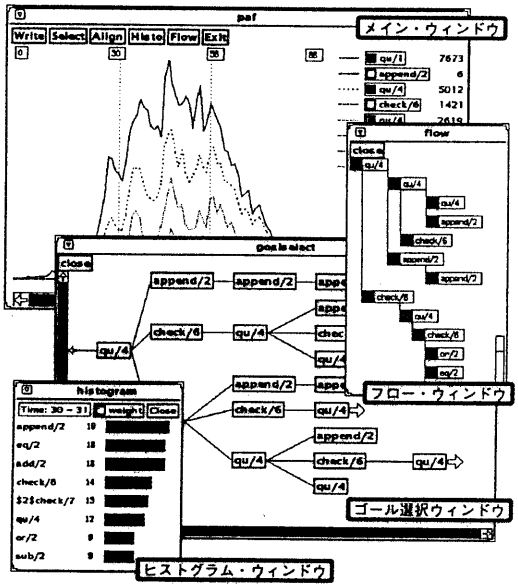

図 4: Paf のディスプレイ

メイン・ウィンドウ

Paf を起動するとまずこのウィンドウが表示され、並列度の表示をおこなう (図 5)。メイン・ウィンドウは、並列度表示部、表示選択部、メニュー部で構成される。並列度表示部には全体および sub tree の並列度が表示される。表示選択部では、後述する方法で選択されたゴール以下の sub tree の並列度を、表示するか否かの選択を行なう。また、そのゴール以下の処理時間 (並列度表示部の面積に相当する) を表示する。並列度表示部でマウスボタンをクリックすると、マーカーを設定できる。マーカーは表示範囲の選択などに使用する。マーカーを 2 つ設定したところでメニューの "Align" ボタンをクリックすると、マーカーではさまれた範囲が並列度表示部に設定される。

並列度表示により全体のおおまかな挙動を観察でき、また sub tree の並列度表示によりプログラムの部分的な挙動を知ることができるので、プログラムの各部分の動作の関連を発見することができる。

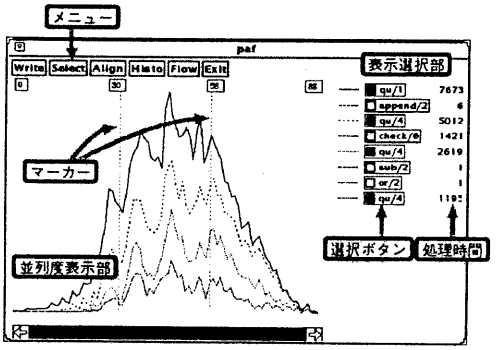

図 5: メイン・ウィンドウ

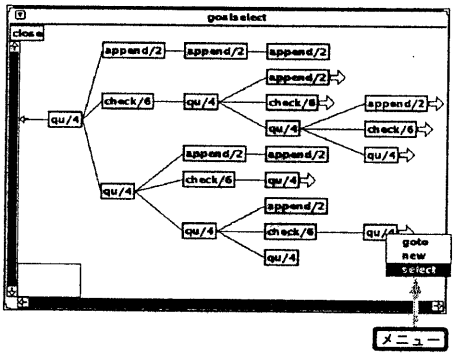

図 6: ゴール選択ウィンドウ

ゴール選択ウィンドウ

ゴール選択ウィンドウは、ゴール・リダクションの tree を表示する (図 6)。ウィンドウには "述語名/アリティ" で示されるゴールが tree 状に表示される。これをたどることで並列度を表示させたいゴールに行きつくことができる。ゴールを示すノードをクリックすると、"goto","new","select" というメニューが表示される。"goto" を選択するとそのノードをルートとして tree を再表示し、"new" を選択するとそのノードをルートとした新しいウィンドウを作成する。"select" を選択すると、そのノードに対応するスイッチがメイン・ウィンドウの表示選択部に登録され、そのゴール以下の sub tree の並列度が表示される。

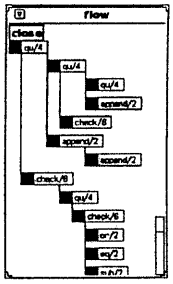

図 7: フロー・ウィンドウ

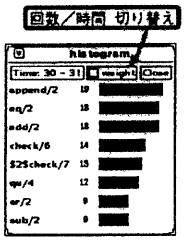

図 8: ヒストグラム・ウィンドウ

フロー・ウィンドウ

ゴール・リダクションの様子を、時刻を反映した形式で表示する (図 7)。並列度の低下している部分での挙動を細かく視覚化することができ、ボトル・ネックとなっている部分の発見に有用である。

ヒストグラム・ウィンドウ

指定した範囲でリダクションされたゴールの数やリダクションに要した時間を表示する (図 8)。ヒストグラム・ウィンドウを使うと、ある時間内で多く実行されたゴール・リダクションを知ることができ、パフォーマンス改善の方針をたてるのに有用である。

6 適用例

Paf を適用した例を示す。

6.1 適用例 1:reverse

リストの内容の順序を反転するプログラム reverse の 2 種類のプログラムと実行フローの様子を図 9 に

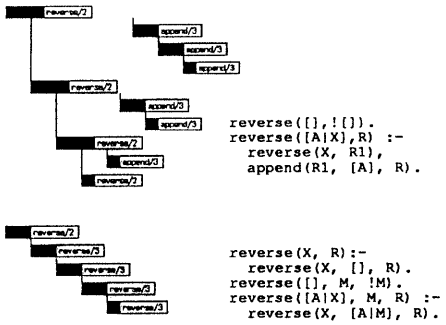

図 9: reverse のフロー

示す。上側が naive-reverse と呼ばれるもので、下側が中間結果を保持して性能を高める工夫をしたものである。naive-reverse の方では append の実行がサスペンドしており、下側のプログラムの方が実行が早く終了しているのが観察できる。

6.2 適用例 2:推論システム

我々の研究室では Fleng で書かれた比較的大規模なアプリケーションとして非単調推論システムを試作している。この推論システムの改良に paf を使用した。

この推論システムは、プロダクション・システムと ATMS(Assumption-Based Truth Maintenance System) で構成されている。しかしプロダクション・システム部は、「照合→競合解消→実行」というサイクルで動作するため、あまり並列度が高くならないということが予想されていた。

そこで paf を使って、推論システムのパフォーマンスを調べてみた。図 10 が全体の並列度である。最初の並列度の低い部分は、ルールを読み込んでコンパイルしている部分であり、並列度の増大している部分 (時刻 38000 付近) から推論が始まる。このグラフでは全体にわたって並列度が脈動し、間欠的にピークが現れるのが観察される。並列度の脈動現象はプロダクション・システム部 (図 11) にも見られる。

競合解消部とルール実行部との関係をグラフ化したものが図 12 である。競合解消部の終了を待ってから、実行部の並列度が上昇しているのが観察でき、競合解消部がパフォーマンスを低下させている原因となっていることが確認できた。

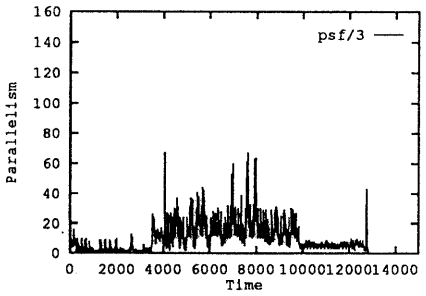

図 10: 改良前の並列度 (実行終了時刻=14823)

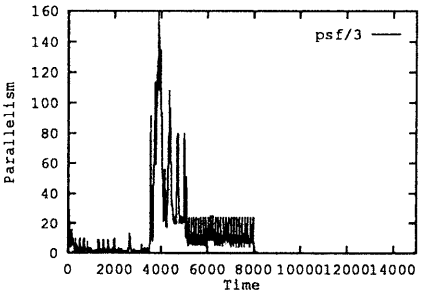

図 13: 改良後の並列度 (実行終了時刻=9706)

図 10 と比べると並列度が増加し終了時刻が早くなっている。

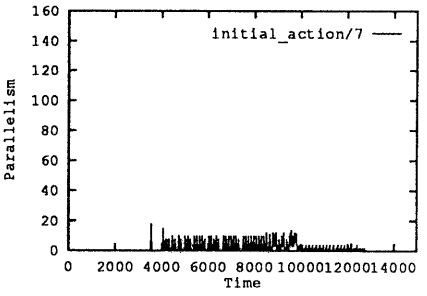

図 11: 改良前の並列度 (プロダクションシステム部)

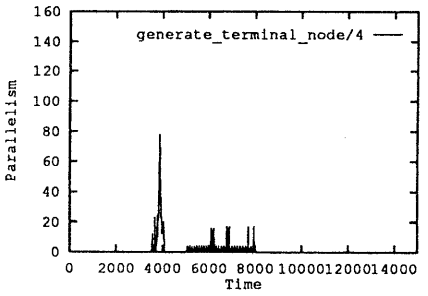

図 14: 改良後の並列度 (プロダクションシステム部)

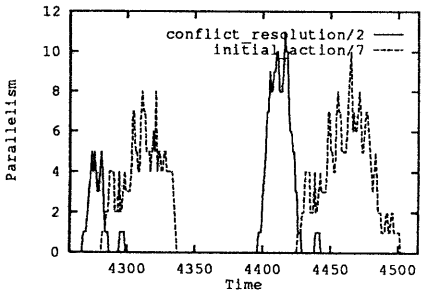

図 12: 競合解消部と実行部の関係

実線が競合解消部、点線が実行部の並列度

このままでは、照合部や ATMS 部の並列性を高めた効果が薄いので、競合解消を行なわずにルールの発火をおこなう、並列発火というメカニズムを導入した。その結果を図 13 に示す。全体の並列度が増加し、実行終了時刻が早くなっている。また、プロダクションシステム部の並列度を図 14 に示す。改良前に見られた脈動現象がなくなっているのが観察できる。

6.3 オーバーヘッド

Paf を使う場合には、pafleng によってトレース・データをとらなければならない。その場合にかかる時間とファイル・サイズを表 1 に示す。トレース・データを記録すると実行時間が 6~12 倍となる。これは、ファイル出力と情報管理処理のオーバーヘッドのため

表 1: トレース・データ記録にかかるコスト

プログラム / リダクション数		実行時間 (ms) データ出力なし	データ出力あり /dev/null	データ出力あり ファイル	ファイルサイズ (KB) 制御情報のみ	ファイルサイズ (KB) 制御情報+束縛情報
ベストパス問題	1,875	67	378	425	47	72
素数計算	11,397	352	2,059	2,321	303	403
推論システム	68,926	2,389	15,973	29,374	2,920	2,127

と、および情報格納領域によりメモリが消費され、ガベージ・コレクションが生じやすくなるためである。

またファイル・サイズは1リダクションあたり31~39バイトとなっている。

7 おわりに

本稿では、細粒度高並列プログラミング言語 Fleng のパフォーマンス・デバッギング・ツール paf の設計と実装、およびその適用例について述べた。

今後の課題としては以下のことを検討している。

データ・フローの視覚化 データ依存関係を出力する機能は既に pafleng に実装してあるので、その情報を視覚化する機能を paf に追加する。また、データ依存関係がパフォーマンスに与えている影響を、プログラマが把握するのを容易にする視覚化を考察する。

システム・パラメータの導入 アルゴリズムの問題を解決した後、プログラム実行環境固有のパラメータ—プロセッサ数, スケジューリング戦略, 負荷分散戦略など— を考慮に入れたパフォーマンスを解析するためのメカニズムを考察する。

参考文献

[1] AIKAWA, S., KAMIKO, M., KUBO, H., MATSUZAWA, F., AND CHIKAYAMA, T.: ParaGraph: A Graphical Tuning Tool for Multiprocessor Systems, in ICOT, ed., Proceedings of The International Conference on Fifth Generation Computer Systems 1992, Tokyo(June 1992), 286–293.

[2] BAGNALL LINDEN, L.: Parallel Program Visualization Using ParVis, in SIMMONS, M. AND KOSKELA, R. eds., Performance Instrumentation and Visualization, chapter 11, 157–187, Addison-Wesley Publishing Company(1990).

[3] HALSTEAD, R. H., JR. AND KRANZ, D. A.: A Replay Mechanism for Mostly Functional Parallel Programs, Technical Report CRL 90/6, DEC Cambridge Research Lab(Nov. 1990).

[4] NILSSON, M. AND TANAKA, H.: FLENG Prolog – Turning Supercomputers into Prolog Machines, in Proceedings of Logic Programming Conference '86, Tokyo(June 1986).