1P−3

細粒度高並列プログラムの実行の視覚化

舘村 純一, 小池 汎平, 田中 英彦
{tatemura,koike,tanaka}@mtl.t.u-tokyo.ac.jp
東京大学工学部*

1 はじめに

予期せぬ動作をするプログラムをデバッグする場合には、まず実行の様子を把握することが必要である。特に細粒度で高並列なプログラムにおいては実行の流れが多数あるので、どこでどのようなことが起きているのか、プログラムの実行の巨視的な状態をまず理解することがより重要な課題になる。このためには、実行情報を抽象化したグローバルな視野が必要となり、デバッガが実行情報をユーザにいかに見せるかといった、プログラム実行の視覚化手法の問題を解決しなければならない。

プログラムの実行の視覚化として研究されているものには、(1) ビジュアルデバッガ (2) アルゴリズム・アニメーションがある。従来のビジュアルデバッガはプログラム言語レベルの抽象度の図形を用いるが、細粒度で高並列なプログラムでは大規模で複雑になり、理解が困難である。一方アルゴリズム・アニメーションは、設計・アルゴリズムレベルの抽象度の図形でプログラムの動作を表現する。しかし、(1) 仕様記述の手間がかかり、仕様中にもバグが存在しうる (2) 予期しない動作の視覚化が重要である (3) バグに到達するにはより低レベルなビューも必要であるなどの理由でデバッグには適さない。

アルゴリズム・アニメーションのような高レベルの視覚化はソース以外の情報なしでは実現が難しい。しかもデバッグ時には、同じプログラムでもどの部分をどのように見たいかというユーザの意図が状況によって変化するので、ユーザの主観を反映させて「見たいものを見たいように見せる」ことが重要となる。我々のデバッガでは「付加的な知識」を与えることによって、プログラム全部について完全な知識を与えなくても、知識を与えない部分は低レベルな視覚化でサポートし、知識の与え方に応じて高レベルなデバッグを可能にする。

本研究では、並列論理型言語の一種である Committed-Choice 型言語 (CCL) を対象とする。CCL は、個々のゴールが並列実行される細粒度並列言語であり、いくつかのプロセスが静的に存在してデータを介して相互作用をし合うのではなく、ゴールは動的に生成・消滅していく。このために、ユーザがゴールの実行を観察したり操作したりするのは困難である。本稿では、Committed-Choice 型言語 Fleng のデバッガ HyperDEBU におけるプログラム実行の視覚化機能について述べる。

2 マルチウインドウデバッガ HyperDEBU

我々は、Committed-Choice 型言語 Fleng のデバッガとして、多次元的インタフェースを用いたマルチウインドウデバッガ HyperDEBU を開発した [1]。このデバッガは、制御・データの流れが形成する複雑なグラフ構造を観察・操作するための多様な視野としてウインドウを提供する。

HyperDEBU は、(1) プログラムの実行をグローバルに観察操作するトップレベルウインドウ、(2) 任意のプロセスに割り当てられるプロセスウインドウ、(3) 構造データを観察するストラクチャウインドウから構成されおり、以下にあげる各機能が協調してユーザのバグ探索を支援する。

1. **多様な視野を用いたバグの絞り込み**：HyperDEBU は、グローバルな視野からよりローカルな視野までをユーザに提供する。プロセスウインドウは、トップレベルウインドウに表示されている各プロセスから開くことができ、多様な側面からプロセスを観察・操作する。ここからさらにサブプロセスを別のウインドウとして開くことで、効果的なバグの絞り込みが行なわれる。
2. **プログラム実行の視覚化**：トップレベルウインドウ上でグローバルなビューとして実現され、実行状況の把握を支援することで、バグの絞り込みを効率化している。
3. **ブレークポイント**：HyperDEBU では、「ブレークポイント」を拡張して考え、デバッガがユーザから実行前に予め与えられた知識ととらえる。この情報はプログラムの実行制御・実行の視覚化などに活用される。
4. **プログラム・コードのブラウジング**：ブレークポイントの設定時などで静的情報の把握を支援する。

3 HyperDEBU における実行の視覚化

高並列プログラムの視覚化は、多数の制御の流れ (ゴールリダクション) とデータの流れ (ガード＋ユニフィケーション) を扱う。それぞれの履歴情報はプロセスウインドウで観察できるが、実行状況の巨視的な把握には繁雑過ぎる。そこで、それぞれについてユーザの主観を反映させた抽象化を行いプログラムの挙動を視覚化する。また、CCL では動的にデータ・ゴールが生成されるので動的な視覚化 (配置) 手法が必要となる。

3.1 制御の流れの視覚化

トップレベルウインドウは、特定のゴールに関するプロセスのみを表示することで制御の流れに関するグローバルな視野を提供する。図 1 はトップレベルウインドウの表示である。

各プロセスは、ウインドウの中に表示された矩形で表現される。これらの表示は、実行状態を反映して動的に変更され、プロセスの生成や状態変化、引数のデータの変化が把握できる。また、矩形からプロセスウインドウをとり出して、より詳しい観察ができる。

*Visualization of Fine-grained Highly Parallel Programs
Junichi TATEMURA, Hanpei KOIKE, Hidehiko TANAKA,
the University of Tokyo

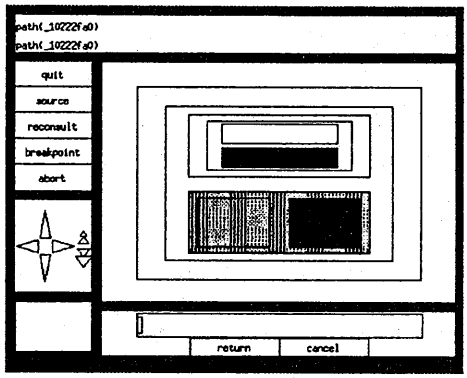

図 1: トップレベルウインドウ

ブレークポイント　ユーザがブレークポイントで指定したゴールに関するプロセスのみ矩形で表示し、他は内部のゴールとして抽象化されて、制御の流れの概略が観察できる。場所は述語名単位、定義節単位、ボディーゴール単位で指定できる。

3.2　データの流れの視覚化

データの流れをグローバルに観察するため、視覚化されたプロセス間に存在するストリーム通信に着目する。そこで、ストリームが生成される様子、それが分配される様子、データの入出力が行なわれる様子を図2のように視覚化する。ストリームが生成されるのは、ボディーゴールに新たな共有変数が生まれた時である。図 2-(1) はストリームが生成された時に画面に視覚化される図形の様子を表したものである。図 2-(2)(3)(4) はそれぞれゴール c がリダクションされてストリームの分配・出力・入力が行なわれた状態を表す。

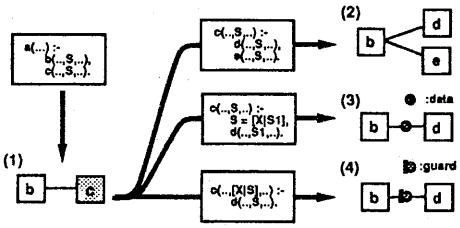

図 2: ストリームの生成・分配・入出力

ブレークポイント　着目するストリームの変数について、各定義節で生成・分配・出力・入力にあたる部分を指定すればよいが、一つ一つ行なうのは繁雑である。これを解消するためにユーザはどの述語のどの引数をストリームとして着目するかだけ指定し、この情報を用いて各定義節のどの部分を視覚化すればよいかを自動的に決定する機能が考えられる。

配置手法　プロセス・ゴール・データをノードとするグラフの配置は以下の手法により動的に行なわれる。

1. ノードのグループ化：互いにストリームでつながれたノードをひとつのグループとし、これに矩形領域を割り当て、制御の流れの視覚化と同様にして配置する。
2. グループ内の配置：グループに割り当てられた矩形領域は、ノードの数だけの矩形に分割され、それぞれの中央に各ノードが配置される。矩形はノードが増えるたびに分割され、各矩形の面積が同じになるように再配置が行なわれる。あるノードが二つのノードに分割されたときは、そのノードのおかれた矩形領域が生成された時の分割面と垂直にその領域を分割する。図 3はノードが生成されていく様子を示したものである。

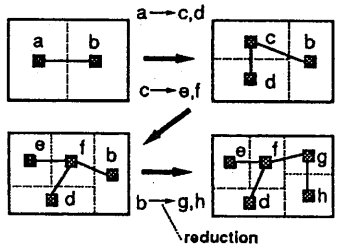

図 3: プロセス・ゴール・データの配置手法

制御の流れの視覚化との融合　ブレークポイントの選択により、制御の流れとデータの流れを融合して視覚化することができる。これにより、ストリーム通信をするものは近くに配置され連結されるので、プロセス間の関係が把握しやすくなる。また、プロセスとして抽象化された矩形内部のデータフローを表示しないことで表示するグラフの繁雑化を避けられる。

表示例　図 4はクイックソートの例である。partition がデータを分配し、qsort がストリームで一列につながって解を収集している様子が視覚化されている。

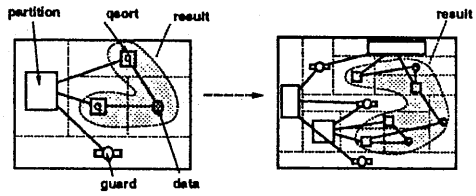

図 4: データの流れの視覚化例

4　おわりに

本稿では、HyperDEBU における細粒度高並列プログラムの視覚化手法について述べた。HyperDEBU は現在アプリケーション開発での試用を通じて評価中であるが、データの流れに関する視覚化は実装を進めている段階である。

参考文献

[1] 舘村, 小池, 田中：並列論理型言語 Fleng のマルチウインドウデバッガ HyperDEBU, JSPP '91, pp. 253-260 (1991).