数理モデル化と問題解決 26－6
（1999．9．22）

データ分割グラフの3次元視覚化

杉田 望代＊　笹倉 万里子†　城 和貴＊
moyo@ics.nara-wu.ac.jp
＊奈良女子大学 理学部　†岡山大学 工学部

概要

本稿では、自動並列化コンパイラにおける統一的中間表現について概説し、その一例であるデータ分割グラフ（DPG）の特徴を紹介する。そして、一般的な視覚化の有効性を述べた後、DPGの視覚化を提案する。DPGのデータアクセス情報は、変数単位でデータが表現され、アクセスの種類も分けられており、詳細な内容を含んでいる。そこで、最適なデータのメモリ配置の支援を目的とする、DPGの変数アクセス情報の視覚化の提案を行う。さらに、そのプロトタイプ実装について報告した後、具体例をもって本提案の有効性を検証する。

Visualization of Data Partitioning Graphs in a 3-D Space

Moyo Sugita＊　Mariko Sasakura†　Kazuki Joe＊
＊Nara Women's University　†Okayama University

Abstract

In this paper, an overview of Universal Intermediate Representations (UIRs) for parallelizing compilers is presented and the characteristic of Data Partitioning Graphs (DPGs) is described. The general effectiveness of visualization has been mentioned,and the visualization of DPGs is proposed.The data access information of DPGs is represented by variable with access property. Then a visualization method of variable access information in DPGs is proposed for the support of optimal data distribution. In addition, a prototype implementation is reported to validate the effectiveness of our proposal by a concrete example.

1　はじめに

並列プログラムの開発形態には、プログラマ自身が直接並列プログラムを記述する場合と自動並列化コンパイラを用いて開発する場合がある。プログラマ自身が並列プログラムを記述する場合、プログラマに対して、並列アルゴリズム、並列プログラミング、対象並列計算機に関する多大な知識と経験が要求される。さらに、特定の並列計算機用に最適化された並列プログラムは、他の並列計算機上で十分な性能を動作する保証がない。

次に、自動並列化コンパイラを用いて開発する場合であるが、面倒な並列プログラミングをプログラマ自身が行う必要がない。従って、プログラマは今までと同じように逐次プログラムを開発するだけでよい。また、過去に逐次プログラム言語で記述されたバグのない膨大なソフトウェア資産を無修正で継承できるのも大きな利点である。さらに、対象並列計算機の違いは、コンパイラが吸収するため、移植の問題が生じない。これらの理由から、自動並列化コンパイラを用いた開発形態が理想的だと考える。

ところが、自動並列化コンパイラは、まだまだ研究対象の段階で、実用にいたっていない。そのため、機能を限定した一部商用のものを除いて、実用に耐えうる汎用の自動並列化コンパイラは、今だ実現されていない。その理由として、以下のことが挙げられる。自動並列化コンパイラは、逐次プログラムを中間表現に変換し、中間表現に並列化を含む種々のプログラム変換手法を適用し、目的プログラムを構成するものである。しかし、さまざまなプログラム変換手法は、各研究者によって独自に研究開発されてきたために、それに伴って、各変換手法が必要とする中間表現が複数提案されてきた。ところが、これらのプログラム変換手法の大部分は互換性に乏し

いため、複数適用する場合、しばしば干渉し合うという問題が生じる。

この問題を解決するためには、コンパイラに各プログラム変換手法が使用する中間表現を統合・共通化した統一的中間表現（UIR : Universal Intermediate Representation）を持たせ、これに対して全てのプログラム変換手法を適用することが必要である。

奈良女子大学、奈良先端技術大学院大学、和歌山大学が共同開発しているNarafrase[1, 2]は、このUIRを用いた分散メモリ型並列処理システムのための自動並列化コンパイラでである。Narafraseではこれ UIRとしてデータ分割グラフ（DPG : Data Partitioning Graph）[3]を採用している。DPGは制御依存とデータ依存、タスクの変数アクセス情報を含んでいる。また、DPGに基づいて、データ及びプログラムの同時分割アルゴリズム（CDP^2）[4]も提案している。

DPGは複数の情報を効果的に組織化しており、今後、統一がはかられるであろう中間表現の有力な候補であると考える。しかしながら、DPGは複数の構成要素から成っており、極めて複雑である。この複雑さは、コンパイラ開発の致命的な妨げにはならないが、開発のターンアラウンドタイムの増加を引き起こし、迅速で柔軟な開発の妨げになりうるものである。従って、複雑なDPGを直感的に把握するための支援システムが期待される。

本稿では、統一的中間表現を人間に直感的に理解させる手法について考察し、その実装に関して報告する。以下第2章では、統一的表現の概説、第3章では視覚化によるDPGの直感的理解の提案、第4章では視覚化のための定義や方針、さらに、プロトタイプを用いた本提案の検証を行う。

2 統一的中間表現の理解

2.1 データ分割グラフ

今日、大規模な並列計算機は、分散メモリ型を採用したものが主流である。しかし、リモート・データ・アクセスあるいはプロセッサ間通信等の通信遅延が生じるため、最適化された並列プログラムを稼動させなければ、実行オーバーヘッドが大きくなるという問題が生じる。そこで、分散メモリ型並列計算機のための自動並列化コンパイラにおいて、データの分割配置ならびに転送の最適化をはかることは重要である。

一般に、データの分割配置ならびに転送の最適化をはかるプログラム変換手法では、個々のタスクの変数アクセス情報、ならびに、アクセス・コスト等の情報が必要である。そのため、依存グラフを中間表現として用いる既成の多くのプログラム変換手法との間で中間表現を共通化し、それらのプログラム変換手法の統合をはかるためには、個々のタスクのアクセスする変数ならびにそのアクセス・コスト等の情報が表現されるような中間表現が望まれる。

そこで、制御フロー、制御依存、タスクの変数アクセス（頻度、データ依存を含む）、ループのネストレベルを同時に表現できるDPGが提案された。DPGは、2種類のnodeと3種類のエッジで構成されている。2種類のnodeは、C-node、D-nodeと呼ばれ、それぞれタスクと変数を表している。また、3種類のエッジのうち2つは、C-node間に存在し、制御フロー、制御依存エッジと呼ばれ、制御の流れ、制御依存を表している。残りのエッジは、データ・アクセス・エッジと呼ばれ、DPG特有のもので、データ依存情報に加え、タスクの変数アクセス情報を表現している。

データ・アクセス・エッジは、C-nodeとD-nodeの間に存在し、方向性がある。そのため、D- nodeからC-nodeと、C-nodeからD-nodeへの2種類のエッジがある。D-nodeからC-nodeへのエッジは、C-nodeタスクによるD-node変数集合に属する変数へのReadアクセスを意味し、C-nodeからD-nodeへのエッジは、C-nodeタスクによるD-nodeの変数集合に属する変数へのWriteアクセスを意味する。あるD-nodeが複数のC-nodeとWriteアクセスを含むデータ・アクセス・エッジで結ばれている場合、それらはデータ依存を表していることになる。詳細は文献[3]を参照していただきたい。

2.2 変数アクセス情報の直感的な理解

このように、DPGは多くの情報を含み、また、大きなプログラムと照応されることが多く、非常に複雑である。さらにDPGは、Narafraseでクラス・オブジェクトとして実装されているため、コンパイラ開発者がDPGを参照する場合、テキスト表示に変換しなければならない。そのため、人間がDPGを理解したり、特徴を把握することは、非常に困難である。そこで、我々はDPGの理解を支援するために、DPGの視覚化を検討した。

人間の視覚認識能力が優れていることは、広く知られている。現に、人間の網膜内の視神経は約1億

3 千万個存在し、光刺激を受け、その光刺激が網膜を出るときには、約 100 万本の視神経繊維の束に整理されて、大脳に伝達される。大脳では、視覚情報を直接処理する細胞は少なくとも 20 億個あり、大脳全体の約 5 分の 1 を占めている。このことからも、いかに人間が視覚認識能力に優れているということが、生理学的視点からも伺い知ることができる。このような人間の認識特徴を生かし、視覚化を用いて DPG の理解支援システムを作る。

DPG は、中間表現の統一化をはかるためにさまざまな構成要素からなるが、人間は、フローやアクセスなどの関連性を追いながら、1 つずつ解析し、筋道立てて理解を深めていく。そのため、構成要素が多く、各構成要素間の関連性が大きい DPG を論理的に理解することは、非常に困難である。そこで、上で述べたように、直感的な理解を促す視覚化を行う。最適なデータのメモリ配置を行うには、タスクの変数アクセス情報を必要とするので、変数アクセス情報のみを抽出し視覚化すると、直感的により理解しやすいヒューマン・インターフェイスが構築できる。本稿では、視覚化の提案だけでなく、実際にプロトタイプを実装し、その効果の検証を行った。

3 視覚化の設計と実装

3.1 視覚化の目的と要件

まず、データのメモリ配置に必要な、各タスクと変数、変数の大きさ、タスクの変数アクセスの有無と頻度を取り出す。ここで注意したいのが、時間情報は抽出しないことである。各タスクがアクセスする変数は異なるため、各時間によって最適なデータのメモリ配置は異なる。しかし、各時間で最適なメモリ配置を行っていたのでは、プログラム全体を通してみると、コストが大きすぎるため最適なメモリ配置とは言えない。そこで、本稿では、各時間ごとのメモリ配置を最適化するのではなく、プログラム全体を通して、データのメモリ配置の最適化をはかる。そのため、時間情報は必要ないのである。

3.2 視覚化の方針

抽出した情報を数値データから幾何学的なデータへ変換する。まず、複数の小さな球と 1 つの大きな球を用いて、C-node と D-node を表す。C-node を表す小さな球は、それぞれある規則に従って色をつけ、大きな球に入れる。大きな球の球面には、D-node を貼る。各 D-node が区別できるように、ある規則に従って色分けし、D-node の大きさに応じて、貼る面積を調節する。D-node の貼りつけパターン, つまり、データのメモリ配置のパターンは NP-complete 問題である。そのため、貼りつけ方の自由度を高め、並列計算機のトポロジー間の差異をうまく吸収できる、球面を選んだ。次に、タスクの変数アクセス情報を加える。アクセスのある C-node、D-node 間は線で結び、アクセス頻度を引力として表す。Read アクセスエッジは赤、Write アクセスエッジは緑で描画される。

3.3 実装環境と視覚化の例

実装には、Windows98 上で VisualC++ と OpenGL1.1 を用いた。

視覚化の一例として、図 1 を挙げる。これは、Livermore ベンチマークカーネル 10 番のループボディ (図 3.3) について DPG をつくり、それを視覚化したものである。球と線という単純なオブジェクトで構成し、視覚的に把握しやすいものに仕上がっている。アクセス頻度は、C-node と D-node との間に働く引力で表しているため、タスクの位置からアクセス頻度を観測でき、目に見えないアクセス頻度を直感的に理解することが容易である。また、アクセスの有無は C-node と D-node を結ぶ線で表現され、アクセスの頻度を表す引力も考慮にいれると、C-node の位置から、現在のメモリ配置の評価ができる。

4 結論

本稿では、分散メモリ型並列計算機におけるデータのメモリ配置問題を、新たなる視点で考察するために、コンパイラの統一的中間表現の 1 つである DPG を視覚化することを提案した。また、当該視覚化システムのプロトタイプを開発し、実際の視覚化の検討を行った。その結果、現段階でのメモリ配置状況を、直感的に把握・評価できることが判明した。また、本視覚化システムは、球と円盤、線といった単純な形のオブジェクトで構成要素とし、視覚的にもとらえやすいものになった。実装は Windows98 上で行ったが、UNIX 環境での利用も鑑みて、OpenGL を採用した。

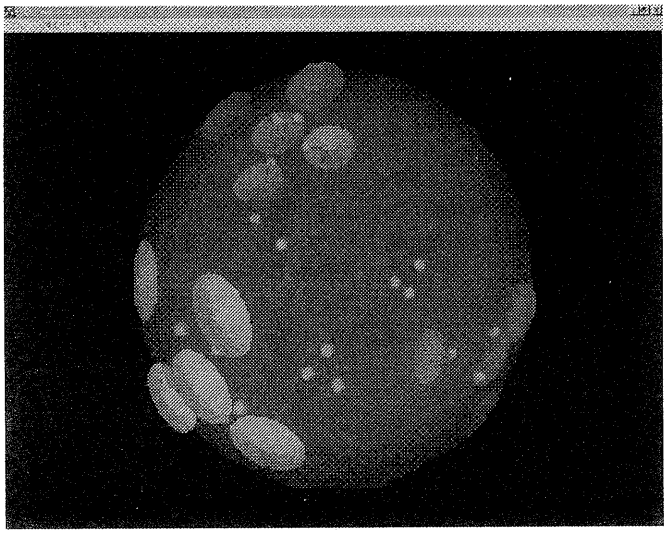

図 1: DPG の視覚化例

```
AR     = CX(5)
BR     = AR-PX(5)
PX(5)  = AR
CR     = BR-PX(6)
PX(6)  = BR
AR     = CR-PX(7)
PX(7)  = CR
BR     = AR-PX(8)
PX(8)  = AR
CR     = BR-PX(9)
PX(9)  = BR
AR     = CR-PX(10)
PX(10)= CR
BR     = AR-PX(11)
PX(11)= AR
CR     = BR-PX(12)
PX(12)= BR
PX(14)= CR-PX(13)
PX(13)= CR
```

図 2: Livermore
ベンチマークカーネル

今後、本視覚化システムの改良のために、新しい機能についても検討をしている。本視覚化システムの現状では、D-node を固定し、アクセス頻度を示す引力によって、C-node が平衡となる位置まで移動するようになっているが、D-node も可動にすることを考えている。D-node を可動にすると、引力に従って C-node と D-node が動き、平衡状態に達する。平衡状態では引力の働く C-node、D-node 同士が集まり、最適なデータのメモリ配置を示唆する状態になる。ユーザは、ある程度グループ化した C-node と D-node の明示的な分割を行い、最適に近いメモリ配置が完了する。また、NaraView（並列化支援視覚化システム）[5] に本視覚化システムを組み込み、並列化の総合的な支援環境を整えたい。

参考文献

[1] K.Kambe, T.Nakanishi, K.Joe, Y.Kunieda, F.Kako: *An Implementation of Loop Transformations with a Universal Intermediate Representation Interface Library*, PDPTA'99, pp.1905–1911 (1999).

[2] 羽田昌代、神戸和子、中西恒夫、城和貴：自動並列化コンパイラの統一的中間表現とインターフェースを用いたコード変換の実装, 情報処理学会研究報告、ARC-134-26,pp.151–156 (1999).

[3] T.Nakanishi, K.Joe, H.Saito, C.Polychronopoulos, A.Fukuda, K.Araki: *The Data Partitioning Graph: Extending Data and Control Dependencies for Data Partitioning*, LCPC, pp.170–185 (1994).

[4] T.Nakanishi, K.Joe, H.Saito, A.Fukuda, K.Araki: *The CDP^2 Algorithm: A Combined Data and Program Partitioning Algorithm on the Data Partitioning Graph*, LCPC, pp.170–185 (1994).

[5] M.Sasakura, K.Joe, Y.Kunieda, and K.Araki: *NaraView: An Interactive 3D Visualization System for Parallelization of Programs*, Int'l J. of Parallel Programing, Plenum Pub., Vol. 27, No. 2, pp.111–129 (1999)

[6] *OpenGL Reference Manual* , Addison-Wesley, 2nd edition (1995).