

形式化に基づく並列性抽出 - 既存逐次プログラムを並列実行するためのパラダイム

笹倉万里子 中西恒夫 城 和貴 荒木啓二郎
奈良先端科学技術大学院大学

人が膨大なデータの関係把握するための一つの有効な手段としてデータの視覚化がある。ここでは、逐次プログラムを並列実行するという問題を例としてデータを視覚化するための一般的な枠組を提案し、実現のための課題を述べる。この枠組では、対象は内部的には複数のモデルで形式的に記述される。また、視覚化情報を視覚化のためのモデルとして形式的に記述しておく。形式的に記述されたデータと視覚化技法の対応をとることで視覚化が行なえる。モデルとしては等価だが見かけの違う視覚化情報を対応させることで、あるデータから異なった図を簡単に得ることができるであろう。

Visualizing data based on formal description – for extracting parallelism in serial programs

Mariko Sasakura, Tsuneo Nakanishi, Kazuki Joe and Keijiro Araki
Nara Institute of Science and Technology

Visualization is one of the useful methods for grasping relations of large data. In this paper, we propose a general model for visualization and discuss problems to realize the model dealing with a problem of extracting parallelism in serial programs. In this model, we describe data and visual informations formally. We consider visualization as matching between data and visual informations. We can get various picture just by changing parameters of the matching.

1 はじめに

人と人とのコミュニケーションを円滑に行なうための一つの方法として図を効果的に使うとよいことが経験的に知られている。例えば、ソフトウェアなどのマニュアルの作成においても文章と図を効果的に組み合わせることが推奨されている。

文章がものごとを論理的に「正しく」記述することに適しているのに対し、図はものごとを直観的に理解することに適している。特に元となるデータのすべてを図示するのではなく、いくつかの属性同士の関係に着目して図示した場合にその特徴が顕著に現れる。

しかしこのことは、逆に図の欠点であるともいえる。特に元のデータが膨大であるような場合、わかりやすい図を一つ見ているだけでは全体を理解することは難しい。そのような場合には、全体に対する理解を深めるためにいろいろな属性の組合せによる図を見ることが有効であると考えられる。これは現在でも行なわれていることで、例えばある統計を取った時に、それを年齢別に表示するグラフを作ったり、地域別に表示するグラフを作ったり、統計を取った年の順に表示して推移を見たりすることに相当する。

統計的データの視覚化に関しては経験的に図表が適していると考えられている。実際にあるデータから属性を指定して簡単に図表を描くツールはすでに存在し実際に使われている。

ここで我々は、統計的データのグラフ表示ツールが行なっていることを一般化することを考える。すなわち、一般的なデータを視覚化するためのツールの構築である。我々はこれを対象データを形式的に記述し、その形式的記述を視覚化することで実現することを考える。ここでは逐次プログラムの並列性抽出を例にしてその意義と課題を述べる。

2 節ではこのような一般的なツールが持つべき枠組について述べる。3 節で、ここで例として用いる逐次プログラムの並列化に関する知識を説明し実際に簡単な例でその視覚化を試みることでその効果を示す。4 節で関連する研究に言及し、最後にこれからの課題と研究の方向を 5 節で述べる。

2 汎用視覚化ツールの基本的な枠組

この節では一般的なデータを視覚化するためのツール – これをここでは汎用視覚化ツールと呼ぶことにする – の基本的な枠組について述べる。図 1 にこの枠組の概念図を示す。

このツールは対象データ、対象の形式的記述、形式的に記述された視覚化のためのモデル、実際の図、の 4 種類のデータを扱う。このツールの特徴は対象の形式的記述と視覚化のためのモデルがそれぞれ複数存在することである。

ここでいう視覚化のためのモデルは具体的には視覚化のために実際に画面上に現れる図の部品となるグラフィカルオブジェクトと、そのグラフィカルオブジェクトの配置の仕方を決める視覚化パターンからなる。グラフィカルオブジェクトとしては例えば直線、長方形、円といった基本的なものからそれらを組み合わせた複雑なものまでさまざまなものが考えられる。視覚パターンは例えば木構造、円グラフ、有向グラフ、あるいは位置関係でグラフィカルオブジェクト同士の関係を表すものなど、さまざまなパターンが考えられる。これらを形式的に記述することにより、グラフィカルオブジェクトや視覚化パターンについてのモデルが明確になり、モデル自体として異なるもの、モデルとしては等価だがみかけが異なるものを区別することができる。

一方対象データを形式的に記述するということは、あるモデルに基づいて対象を抽象化しその本質を記述することである。したがって、異なったモデルを用いれば異なった形式記述が得られる。ここで得られる複数の形式記述はどれも同じ対象を表していると同時に、それぞれのモデルを明確に表していると考えられる。

これらの複数の形式的に記述された視覚化のためのモデルと形式的に記述された対象のモデルを比べることによって、ある対象モデルとある視覚化モデルが似ているかどうかを形式的に判断することが可能になるかもしれない。つまり、ある対象モデルを視覚化するのに適した視覚化のためのモデルと適さないモデルを判別することが可能にあると思われる。

これにより、ある一つの対象に対して異なったモデル化を行なうことで最終的に異なる図を出すこと

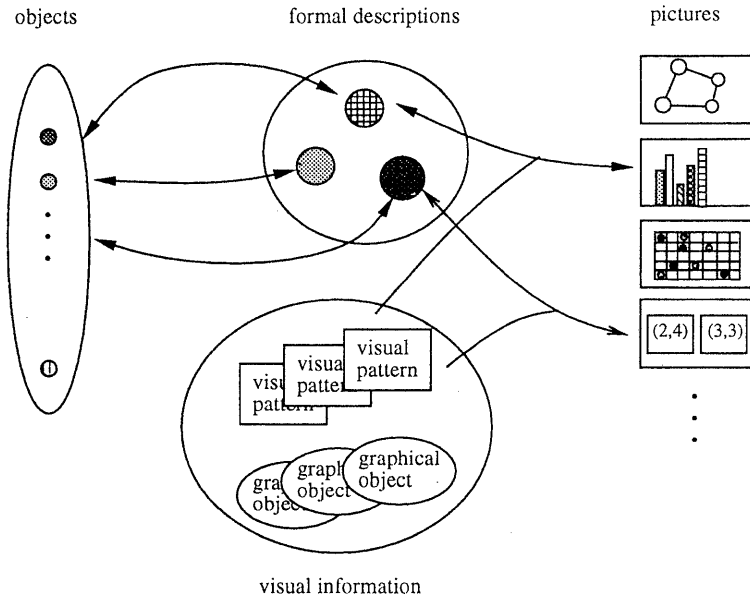


図 1: 汎用視覚化ツールの枠組の概念図。対象を形式的に記述し、視覚化情報(視覚化パターン、グラフィカルオブジェクト)との対応をとることにより図を生成する。

ができる。また、同じモデルでも見かけの違うグラフィカルオブジェクトや視覚化パターンを適用することで見かけの異なる図を表示することができる。つまり、ユーザはある一つの対象のさまざまなモデルをいろいろな観点から視覚化してみることが可能になる。これは特に膨大なデータを直観的に理解するには有用であると考えられる。また、表示の仕方を変えることで全く違う図が得られ、そこから思わぬ発見が起きることも考えられる。ここから逆にモデル形成に示唆を与えることも期待できよう。

以降では例として逐次プログラムの並列化を対象としてこのツールの効果を示す。

3 並列化抽出を目的とした逐次プログラムの視覚化

この節では、図を利用して簡単な逐次プログラムのループを並列化する問題を考える。3.1 節ではこの例を理解するために簡単に並列プログラミング・モデルについて説明する。3.2 節では具体的な視覚化の

例を示す。

3.1 並列プログラミング・モデル

並列プログラミング・モデルには、共有変数、メッセージ・パッシング、データパラレル、オブジェクト指向方式、論理型および関数型による各モデルがある [5]。ここでは逐次プログラムの並列実行という視点を取るために、並列プログラミング・モデルとしては共有変数モデルを、実行環境としては共有メモリ型並列計算機アーキテクチャを考える。また、逐次型プログラムから並列プログラムへの変換は、プログラマによるものとコンパイラによるものが考えられるが、ここではコンパイラによるものを仮定する。すなわち、与えられた逐次プログラムを特定の並列計算機で実行するために、自動並列化コンパイラを適用するとする。

自動並列化コンパイラとは、逐次プログラムを入力して並列プログラムを出力するコンパイラのことであり、共有メモリ型並列計算機用に関しては、研究用 [14] 商用 [15] 共に開発が進められている。一

般に、自動並列化コンパイラの処理手順は、1) フロー分析、2) 最適化、3) 並列コード生成である¹。フロー分析においては、データ依存 [1] 関係や制御依存 [4] 関係の分析を行なう。これらの目的は、依存関係のある逐次プログラムのセマンティクスを保ったまま依存関係のないプログラムに変形し、並列化を図ることである。例えば、ループ・ボディ内の二つの命令文が互いに依存関係を持つ場合、単純縮小、TD 縮小、選択的縮小と呼ばれる手法を適用して、依存関係を除去できる場合もある。このようなフロー分析の後に適用される最適化では、主に並列性の抽出が行なわれる。例えば、帰納的変数の置換、ループ・アンローリング、定数伝搬、ループ・インターチェンジ、ノード分解、ループ・ブロッキング、ループ・スプレディング、ループ融合等、実に多くの手法が提案、利用されている。詳しくは文献 [13] を参照されたい。このようにして並列性が抽出された中間表現に対して、適当なスケジューリング手法を選び、実際の並列コードを生成する。

逐次プログラムにおける依存関係の表現や、並列性抽出後のプログラムの中間表現には、タスク・グラフがよく使われる。タスク・グラフとは、プログラムの基本ブロックをノードで、依存関係を有向エッジで表した、自動並列化コンパイラの中核をなすデータ構造である。通常、この中間表現に対して、プロセッサ割当等を施し、並列実行可能なプログラムを再構築する。タスク・グラフの依存関係が制御を表すものは制御依存グラフ、データ依存を表すものはデータ依存グラフ、さらに両者を統合して階層的な表現を実現したものが階層的タスク・グラフ (HTG: Hierarchical Task Graph) と呼ばれる [4]。

さて、自動並列化コンパイラは並列性を自動的に抽出することを目標としているわけだが、実際には様々な要因がありどの戦略を適用するのがいいのかを完全に自動的に判断することは難しい。そこで、プログラマにどんな並列化手法をどんな順番で適用して並列化するかを判断を仰ぐことになる。しかし、複雑なループになるとソースコードからではどのような依存関係があるかを把握するのさえ難しい。

そこで、これらの依存関係を把握しやすくするためにループ情報を視覚化することを考える。もし、複

```
DO I = 3, N1
  DO J = 5, N2
    ...
    A(I,J) = B(I-3, J-5)    ---S1
    B(I,J) = A(I-2, J-4)    ---S2
    ...
  ENDO
ENDO
```

図 2: 逐次プログラムの例

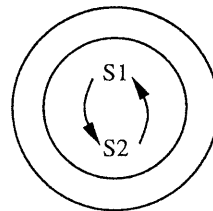


図 3: 図 2 のタスクグラフ。矢印は依存関係を示し、円はループを示している。

雑なループを適切に視覚化することができれば、人がそこから並列性を読みとり易くなるかも知れない。それが可能となれば、自動並列化コンパイラ研究はもとより、並列プログラミングそのものに与える影響も大きいものであろう。

3.2 視覚化の例

ここでは、[13] で使われている図 2 のループを使って視覚化の例を示す。ここではループの依存関係を調べて並列可能性を検討することが目的である。そこで、ループ内の文の依存関係、実行順序に基づいてループをモデル化したと仮定し、その上でどのように異なった視覚化が行なえるかを示す。我々の目標は汎用視覚化ツールでこのようなさまざまな図を簡単に作りだせるようにすることである。

まず、ソース上の文を単位として依存関係を表示した例が図 3 である。これは一般にタスクグラフと呼ばれている。この図を見るときもつと内側のループの中で S1 から S2 への矢印と、S2 から S1 への矢印が存在していることから S1, S2 には双方向に依存関係があり、このままでは並列化できないことが

¹ここで紹介するのは自動並列化コンパイラのフロントエンド部分である。実際には、マシンに依存した命令を出力するバックエンド部分が必要である。

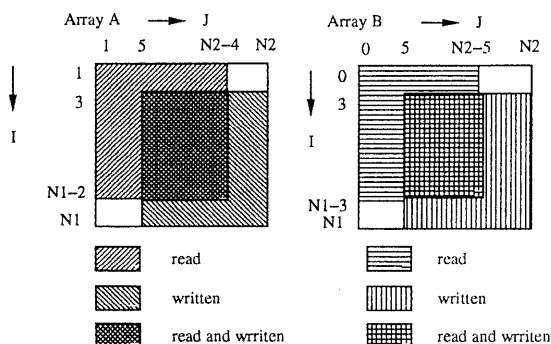


図 4: 図 2 の配列データの参照関係を示した図.

わかる. このループが並列化できるかどうかは他のパラメタを見てみなければわからない.

例えば, ループが実行されることによって, 配列 A,B のどの要素が参照されるかを表示することができる. これを図 4 に示す. これを見ると A,B ともに読み込みだけのために参照されているデータ, 書き込みだけのために参照されているデータが存在していることがわかる. 実際にはこの図からは並列性に関しては情報を得ることができない. なぜならこの図ではどの時点でどの変数が参照されているかわからないからである. しかし, このような図はデータ

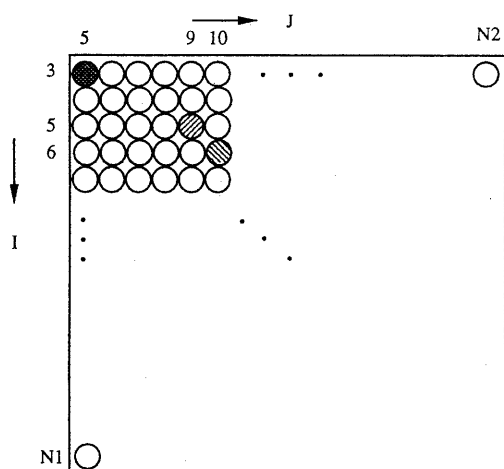


図 5: 図 2 のループを展開して依存関係を示す図.

```

DO I1 = 3, N1, 2
  DOALL I = I1, I1 + 1
    DOALL J = 5, N2
      ...
      A(I,J) = B(I-3, J-5)
      B(I,J) = A(I-2, J-4)
      ...
    ENDOALL
  ENDO
END

```

図 6: 逐次プログラムの並列化その 1

```

DO J1 = 5, N2, 4
  DOALL J = J1, J1 + 3
    DOALL I = 3, N1
      ...
      A(I,J) = B(I-3, J-5)
      B(I,J) = A(I-2, J-4)
      ...
    ENDOALL
  ENDOALL
END

```

図 7: 逐次プログラムの並列化その 2

配置を考える上では重要であると考えられる [10].

ループを展開して具体的に何回目のループの時と何回目のループの時に依存関係が生じるかを図示することも考えられる. その図の一例が図 5 である. ここではすべてのループを展開して一回のループで実行される文をまとめて一つの円としてマッピングしている. 図 5 では $(I,J) = (3,5)$ の時に実行される文と $(I,J) = (5,9)$, $(I,J) = (6,10)$ の時の文が依存関係にあることを示している. 例えばこの図を使ってある (I,J) を指定することによってそれと依存関係にある文を表示することができる. この図でいろいろな文について依存関係を調べると, ある I の行内では (つまり I を固定して J を変えるということ) 依存関係がないことがわかる. また, I 行と $I+1$ 行の間にも依存関係がないことがわかる. このことから図 6 のように並列化することができることがわかる. これは一般に選択的縮小と呼ばれている並列化手法である.

さて, 図 5 をもう一度ながめると, ある J 列につ

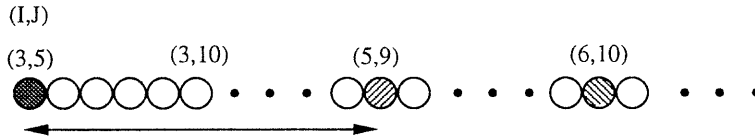


図 8: 図 2 を一元的に展開した場合の図.

いてもやはり依存関係がなく、また、J 列から J+3 列までにも依存関係がないことがわかる。このことから図 7 のように並列化することもできることがわかる。こちらの方が一度に実行できる文の数が多い。しかし、実際にこちらの方が早いかどうかは実行するマシンがどのようなものであるかに依存する。

さて、今度はループの展開後の文の依存関係を別な図で表してみよう。図 8 はループの実行順序その

ままに一元的に展開して文の依存関係を示すものである。これも図 5 と同じようにある文を示す円を選ぶとそれと依存関係のある文を表示するとする。すると、図で矢印で示した範囲では依存関係がなく、これらの文は同時に実行できることがわかる。このことから図 9 のように並列化できることがわかる。これは一般に TD 縮小と呼ばれている手法である。この場合のタスクグラフを図 10 に示す。これを見る

```

DO K = 1, NM, λ
  T1 = (K DIV M) + 3          --S1
  T2 = ((K + λ) DIV M) + 3    --S2
  T3 = K MOD M + 5            --S3
  T4 = ((K + λ - 1) MOD M) + 5 --S4
  DOALL J = T3, M
    ...
    A(T1, J) = B(T1-3, J-5)    --S5
    B(T1, J) = A(T1-2, J-4)    --S6
    ...
  ENDOALL
  DOALL I = T1+1, T2-1
    DOALL J = 5, N2
      ...
      A(I, J) = B(I-3, J-5)    --S7
      B(I, J) = A(I-2, J-4)    --S8
      ...
    ENDOALL
  ENDOALL
  DOALL J = 5, T4
    ...
    A(T2, J) = B(T2-3, J-5)    --S9
    B(T2, J) = A(T2-2, J-4)    --S10
    ...
  ENDOALL
ENDO

```

図 9: 逐次プログラムの並列化その 3

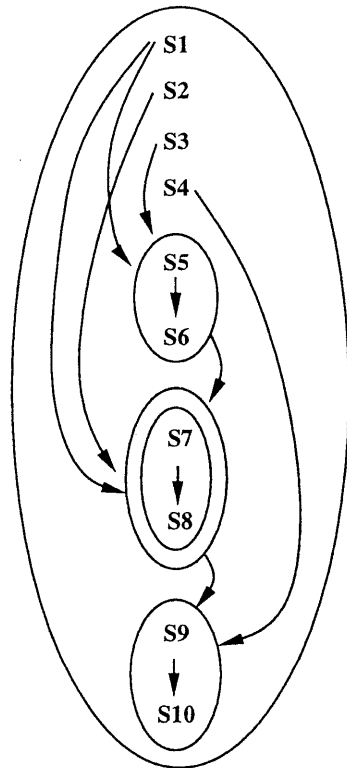


図 10: 図 3 のタスクグラフ

と、一つの内側のループを構成する S5, S6 の間には S5 から S6 への矢印しか存在しないことから、このループは並列に実行できることがわかる。S7, S8 からなるループ, S9, S10 からなるループも同様である。このように図 10 では、図 3 で存在していた並列化できない依存関係が解消され、並列実行可能であることが視覚的に確認できる。

4 関連する研究

visual 言語を形式的に記述しようという試みはいくつかなされている。しかし、その多くはここで扱いたいと考えているものより低い抽象レベルでの形式化を扱っている。例えば、[8] は図の object-type と relation type による形式的記述を試みている。[2] では visual 言語におけるタイプに関する議論がなされている。[12] では簡単な GUI を LOTOS で記述する試みがなされている。

一方、visualization の分野ではある対象に限ってそれを視覚化することが中心となっている。例えば、ここで例としたプログラムの視覚化に関してもいくつかの研究がなされている。[16] は並行プログラムのプログラム支援ツールの一つとして Ada で記述した並行プログラムのタスクグラフの可視化について述べている。[7] はソフトウェアのメンテナンスやリエンジニアリングのためのプログラムの視覚化について報告している。しかし、多くの場合 visualization は対象を理解しやすくするための手段という位置づけである。visualization すること自体を一つのモデルととらえた研究はまだ少ない。[11] は具体的な図レベルでの visualization の形式モデルを述べているが、抽象度の高いレベルについては言及されていない。

形式的記述を行なおうとする上で、visualization を行なう場合と visual 言語を扱う場合では異なるところが問題になると考えられる。例えば、visual 言語では言語としての文法や、ユーザとのインタラクションの扱いが避けて通れない問題として存在する。一方 visualization では、データをどのように図に対応づけるかがもっとも大きな関心事である。しかし、visual 言語においても内容と図の対応関係は無視できる問題ではなく、visualization でも、ユーザとのインタラクションやある文法規則にしたがった合成、推論などが将来的に行なえるようになるのは望まし

いことであると思われる。つまり、現在の visual 言語と visualization では同じ問題を違う側面から見ていると考えることができる。我々は visualization の面からのアプローチを試みているが、将来的には visual 言語を含めることもできると考えている。

5 今後の課題

ここでは、複数のパラメータを持つデータからいろいろな図を表示することの有益性を例によって示し、また、そのための一般的な枠組を提案した。今後逐次プログラムの並列性抽出問題を例としてさらに研究を進めたいと考えている。実際にここで述べた汎用視覚化ツールを実現するためには次のようなさまざまな課題がある。

視覚化のためのモデルを形式的に記述することに関して

4 節で述べたように図の形式的記述はまだもっとも基本的な、線・円レベルで研究されているだけである。汎用視覚化ツールを実現するためにはこれらの基本的なグラフィカルオブジェクトの組合せ、さらに視覚化パターンについての形式的モデルを与えなければならない。

対象の形式的記述に関して 対象を形式的に記述するにはモデルが必要である。逐次プログラムの並列性を記述するためのモデルを構築しなければならない。

対象モデルと視覚化モデルの対応づけに関して あるモデルとあるモデルが何をもって同じか、あるいはどの程度似ているかを判断する基準が必要になるであろう。

具体的な図を生成することに関して ここであげた簡単なループの例だけでもさまざまな課題があることがわかる。例えば、この例ではループ変数が二つしかないので、図 5 のような配列で全体を表現することができるが、変数が 3 以上になったときにどうするかを考えなければならない。また、この例では配列の添字の形が簡単であるために文同士の依存関係が容易に推測できるが、添字がもっと複雑な形であった時、図を見ただけで容易に推測がつくような支援ができるかどうかという問題がある。さらに、一般にループを展開すると膨大な文が出ると考えら

れる。ここでは図8のように適当に省略した形で図示したが、このような「適当な省略」が自動的にできるかどうか問題である。これらには大量のデータを可視化するための研究が役にたつかもしいない [3][6]。

将来的には図を変更したらそれに対応する視覚化モデルの形式的記述が変わり、それにしたがって対象モデルが変化し、最終的に対象そのものが変更されるようにすることも考えられる。例えば [9] では、宣言的に記述されたデータの関係とそれを視覚化したグラフとの間でデータの変更を双方向に行なうことができる。このような機能を一般的に提供することは有用であると考えられる。

参考文献

- [1] Banerjee, U. "Dependence Analysis for Supercomputing", Kluwer Academic Press, 1993.
- [2] Burnett, M.M. "Types and Type Inference in a Visual Programming Language", 1993 IEEE Symposium on Visual Languages, pp.238-243, 1993.
- [3] Furnas, G.W. "Generalized fisheye views", *Proceedings of the ACM Conference of Human Factors in Computing Systems (CHI'86)*, pp.16-23, ACM Press, 1986.
- [4] Girkar, M.B. "Functional Parallelism Theoretical Foundations and Implementations", Ph.D. Thesis, University of Illinois at Urbana-Champaign, 1991.
- [5] Hwang, K. "ADVANCED COMPUTER ARCHITECTURE: Parallelism, Scalability, Programmability", McGraw-Hill, 1993.
- [6] Koike, H. and Yoshihara, H. "Fractal Approaches for Visualizing Huge Hierarchies", 1993 IEEE Symposium on Visual Languages, pp.55-60, 1993.
- [7] Linos, P.K. Aubet, P., Dumas, L., Helleboid, Y., Lejeune, P. and Tulula, P. "Visualizing Program Dependencies: An Experimental Study", *Software - Practice and Experience*, vol.24, no.4, pp.387-403, 1994.
- [8] Meyer, B. "Pictures Depicting Pictures On the Specification of Visual Languages by Visual Grammers", 1992 IEEE Workshop on Visual Languages, pp.41-47, 1992.
- [9] Miyashita, K., Matsuoka, S., Takahashi, S. and Yonezawa, A. "Declarative Programming of Graphical Interfaces by Visual Examples", *UIST'92*, pp.107-116, 1992.
- [10] Nakanishi, T., Joe, K., Saito, H., Polychronopoulos, C.D., Fukuda, A and Araki K. "The Data Partitioning Graph: Extending Data and Control Dependencies for Data Partitioning", submitted.
- [11] Onodera, T. and Kawai, S. "A Formal Model of Visualization in Computer Graphics Systems", *Lecture Notes in Computer Science 421*, Springer-Verlag, 1990
- [12] Paternó, F. "A formal specification of appearance and behavior of visual environments", *Software Engineering Journal(UK)*, vol.8, no.3, pp.154-64, May, 1993.
- [13] Polychronopoulos, C.D. "Parallel Programming and Compilers", Kluwer Academic Press, 1988.
- [14] Polychronopoulos, C.D. et al. "Parafrase-2: An environment for parallelizing, partitioning, synchronizing, and scheduling programs on multiprocessors", *Proceedings of the 1989 International Conference on Parallel Processing*, 1989.
- [15] "FX/Fortran Language Manual", Alliant Computer Corporation, 1986.
- [16] 笠原義晃, 程京徳, 牛島和夫 "Ada 並行処理プログラムにおけるプログラム依存性の可視化", 情報処理学会夏のプログラミングシンポジウム論文集, pp.149-156, 1993.