複合グラフを用いた階層タスクグラフの視覚化

笹倉万里子　木和田智子　城 和貴　荒木啓二郎
奈良先端科学技術大学院大学

逐次プログラムを並列プログラムに書き換える自動並列化コンパイラの分野では，ユーザのプログラムの理解支援，デバッグ，並列性抽出の半自動化のためのインタフェースなどの目的のために並列化支援システムが必要とされてきている．本稿では，イリノイ大で研究開発されている自動並列化コンパイラ Parafrase-2 の中間表現として使われている HTG(Hierarchical Task Graph) を軸としてソースレベルでのプログラムの依存情報を視覚化することで並列化支援を行なうシステムを提案する．特に HTG をグラフとして表示するための方法について述べる．

Visualizing Hierarchical Task Graphs

Mariko Sasakura, Satoko Kiwada, Kazuki Joe and Keijiro Araki
Nara Institute of Science and Technology

For parallelizing compilers which generate parallelized programs from sequential programs, it is important to provide parallelization assist systems which help users to understand programs, debug, and extract parallelism that are hardly found out by only compilers. In this paper, we propose a program visualizing system which is based on HTG(Hierarchical Task Graph). The HTG was designed and implemented as an intermediate expression of the Parafrase-2 which is a parallelizing compiler developed in Ilinois Univ. We discuss an outline of our system and a method of visualizing graphs for HTG.

1 はじめに

自動並列化コンパイラは逐次的に実行されることを念頭において書かれたプログラムを並列プログラムに書き換えるコンパイラである．自動並列化コンパイラを用いると，過去に作成され，使われ続けたバグの少ない逐次プログラムを並列に実行することができる．また，新たにプログラムを作成する時も，複雑な並列アルゴリズムを考えなくても逐次的なプログラムを書けばコンパイラがそのプログラムに内在する並列性をある程度引き出して並列実行することができるので，プログラムの開発コストを低く抑えることができる．

自動並列化コンパイラは逐次プログラムから並列プログラムへの書換えを自動的に行なうことを目的としているが，現時点では完全に自動的に書き換えるのは難しい [16]．プログラムの意味に踏み込まなければ並列性がわからない場合もあるし，実行時にしか決まらない変数が存在するような場合もある．もっとも大きな問題はプログラムのどこの部分をどの並列化手法を使って並列化すれば一番効率良くなるかという判断が自動的にはできない点である．これらの場合にも自動的な並列化が行なえるようにするための研究も進んでいるが，なかなか困難な問題である．そこで，プログラムを視覚化することで並列化支援を行なうことを考える．

昔から大量あるいは複雑なデータ間の関係を理解するのに図を使うとわかりやすいことは経験的に知られている．人間が幾何や物理の問題を解く時には作図して問題解決に役立てていることを実験的に示した研究もある [10]．このようなことから，視覚化は対象に対する理解を深め，新たな発想を支援する手段の一つとして最近注目を集めてきている．

本研究では，並列性を考える上で重要である制御依存とデータ依存をプログラムのソースレベルで視覚化することで並列化支援を行なうことを試みる．このシステムは，

- 自動並列化コンパイラ研究者向けに新たな並列化手法発見のための研究環境を提供する．
- 自動並列化コンパイラを使うユーザに，プログラムのどの部分にどのような並列化手法を適用するかを指示するためのインタフェースを提供する．

という二つの目的を持つ．ここでは，コンパイラとしてイリノイ大学で研究開発された自動並列化コンパイラ Parafrase-2[8] を用いる．Parafrase-2 は現在大学等を中心に広く使われている自動並列化コンパイラのうちの一つで，多くの並列計算に関わる研究が Parafrase-2 をベースとして行なわれている [2],[5]．Parafrase-2 は内部表現として HTG(Hierarchical Task Graph : 階層化タスクグラフ) を用いているのが一つの特徴である．

本稿では，われわれがこれから構築する予定である視覚化システム NaraView の全体構想を述べた後，特に NaraView における HTG のグラフとしての視覚化について述べる．まず，2 節では，Parafrase-2 の内部表現であり，また，われわれがプログラムを視覚化する上での中間表現として用いる HTG について説明する．次に 3 節では，われわれが構築しようとしている並列化支援システムの全体構想について述べる．4 節で自動並列化コンパイラ支援の目的を果たすためのグラフの表示条件を明らかにした後，グラフの表示法について述べる．5 節では関連する研究について言及する．最後に 6 節で今後の課題と展望について述べる．

2 HTG: 階層化タスクグラフ

HTG(Hierarchical Task Graph : 階層化タスクグラフ) は，コンパイラにおける最適化，並列化，コード生成に必要なすべての情報を含んだ一つの中間表現として Girkar らによって設計されたものである．詳しい定義は [4] を見ていただくとして，ここでは HTG を直観的に説明する．

HTG はプログラムの制御フローグラフ (CFG)[1] をループ構造に基づいて階層化したものである．ループまたは条件分岐を一つの単位として階層化する．ループを一つのノードとしてまとめたものをループノード，条件分岐を一つの単位としてまとめたものを複合 (compound) ノードと呼ぶ．ループノード，複合ノードには，その内部の制御フローを表す CFG が含まれる．階層化することによって HTG の任意の階層の CFG は非循環になる．

HTG には，制御フローの情報だけでなく，データ依存関係解析の結果からデータ依存に関する情報も付加される．これは，データ依存関係のあるノード間のアークという形で表現される．

HTG 自体はコンパイラが扱うどの粒度にも対応できるものである．しかし，われわれはプログラムソースの視覚化を目的としているので，ここでの HTG 内

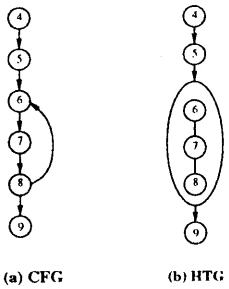

図 1: HTG の一例.

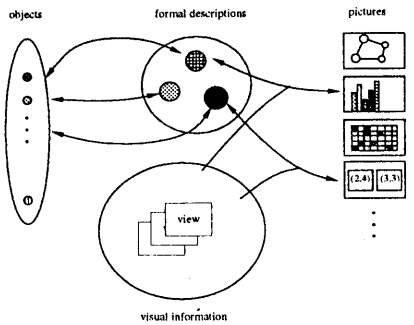

図 2: 汎用視覚化システムの枠組. 対象を形式的に記述し, 視覚化情報 (ビュー) との対応をとることにより図を生成する.

に現れる CFG の一つのノードはソースプログラムの一文に相当する. 図 1に HTG の例を示す.

3 並列化支援視覚化システム NaraView の全体構想

この節では NaraView の設計上の基本方針を述べた後, システムの仕様を簡単に説明する.

3.1 基本方針

NaraView はソースレベルでの依存関係を視覚化することによってプログラムの並列化を支援するシステムである. われわれはすでに人の発想を支援するための汎用視覚化システムの基本的な枠組を提案している [12]. これは, 視覚化しようとする対象を形式的に記述し, それにあらかじめ用意した情報の視覚化表現の方法 (これを以降ではビューと呼ぶことにする) を対応させることによって複数の図を生成しようというものである (図 2). NaraView はこの枠組にのっとって次のように構成される.

視覚化対象 逐次プログラムのソースコード.

視覚化対象の形式的記述 HTG.

ビュー 以下の三種類のビューを提供する.

1. 全体表示.
2. タスクグラフ表示.
3. ソースプログラム表示.

NaraView では複数のビューを用意し, 視覚化対象をいろいろな角度から眺められるようにすることによってユーザの発想を刺激する並列化支援を行なう. その際, 視覚化対象の形式的記述である HTG を各ビューの中間表現とすることでビュー間の対応付けが簡単に行なえるようにする.

3.2 用語の定義

本稿で使用する用語の定義を行なう.

階層 HTG における階層の意味. すなわち, ループのネスト構造に代表されるようなグラフの入れ子構造を階層構造といい, 入れ子の一段一段を階層という.

階層の上下 ネストの浅い方を階層の上の方, ネストの深い方を階層の下の方という.

リアルノード HTG のうち, ソースコードの一文に対応したノード. HTG ではそれ以上細かくできないノードである.

メタノード HTG のうち, ループノードや複合ノードなど, 内部にグラフを含んだノードのこと.

展開/併合 メタノードの内部に含まれているグラフを表示することをメタノードを展開するという. 表示されていたグラフを消してメタノードだけにすることを併合するという.

フェーズツリー メタノードを展開することによって表示されるグラフのこと.

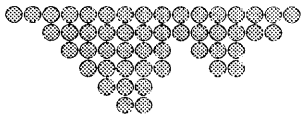

図 3: 全体表示の一例．この例では図の上下方向は階層の深さを示し，左から右へプログラムのながれを示している．プログラムのながれと階層構造が概観できるように大まかに示されている．

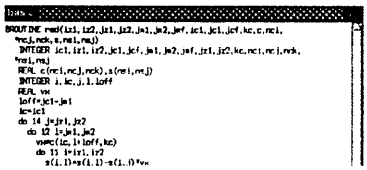

図 4: ソースコード表示の一例．ソースコードの一文は HTG の一つのリアルノードなので，これは HTG のリアルノードをソースでの出現順に並べたものと見ることもできる．

注目ノード ユーザが関心を持っているノードのこと．多くの場合，その時マウスカーソルが指しているノードが注目ノードであると考えられる．

3.3 システムの仕様

入力

NaraView に対する入力は表示対象であるプログラムのソースコードと HTG である．HTG は Parafrase-2 を用いてソースプログラムを解析することで得る．

出力

NaraView の出力はビューである．全体表示，タスクグラフ表示，ソースプログラム表示の三つのビューを提供する．全体表示はユーザがプログラム全体の構造や注目ノードのプログラム上での位置を直観的に把握できるようにすることを目的としたビューである (図 3)．タスクグラフ表示は HTG をそのままグラフとして表示したもので，局所的なプログラムの構造をわかりやすくするためのものである．これについては，4節で述べる．ソースプログラム表示はソースプログラムと全体表示やタスクグラフ表示との対応を見せることで並列化のための情報をユーザに与えることを目的としたものである (図 4)．

ビューにおいて可能な操作

各ビューにおいて表示されているものはそれぞれ HTG の一部でしかない．したがって，ユーザが行なえる操作も各ビューによって異なってくるであろう．ここでは，どのビューにおいても共通と思われるものをあげる．

- メタノードを展開して表示する．逆にフェーズツリーを併合する．
- 全体を拡大縮小する．
- アークの種類ごとに表示／非表示を指定する．
- 注目ノードを指定する．

ビュー間の対応

ビュー間の対応とは，あるビュー A でインタラクティブに行なわれた操作の影響を別なビュー B に及ぼすことをいっている．この対応は中間表現である HTG を経由することで実現される．将来的にはビューにおいて可能なすべての操作を対象とするが，当面は次の機能を対象とする．

- 注目ノードの表示．あるビューで設定された注目ノードが他のビューではどれにあたるかを表示する．
- メタノードの展開，フェーズツリーの併合．インタラクティブにあるビューでメタノードの展開やフェーズツリーの併合が行なわれたらその操作を他のビューにすぐ反映させる方法と，あるビューで独立に展開，併合を何度か行なった後，その結果を他のビューに反映させる方法とがある．

使用例

現段階での NaraView はプログラムの依存関係を視覚化することのみを行ない，インタラクティブにプログラムを変更する機能は備えていない．われわれが想定している NaraView の使用法を示す．

1. **ソースコードを指定して NaraView を起動** これにより NaraView は自動的に Parafrase-2 を起動してコンパイルを行ない，並列化したソースコードと HTG を得る．それをもとに全体表示と一番上の階層のタスクグラフ表示を行なう．

2. **ユーザへの情報提示** ユーザの指示により次のことを行なってプログラムの構造や自動並列化コンパイラによって並列化された部分，並列化されなかった部分の情報を提供する．

   - ソースコードの表示．
   - より下の階層の表示．メタノードを展開することで得られる．
   - タスクグラフ表示とソースコード表示での注目ノードの対応の表示．

3. **ユーザによるソースコードの編集** 提示された情報をもとにユーザはソースコードに対し次のような変更を行なう．

   - 自動並列化できなかった部分を並列化できるようにソースを書き直す．
   - ソースコード上では存在するデータ依存やフロー依存を無視するようにという注釈を書き加える．この場合，再コンパイルするとその依存関係が無視されて並列化される．

4 HTG の視覚化

この節では，HTG のタスクグラフ表示の方法について述べる．HTG をどのようにしてグラフとして表示するかという問題は，グラフの自動描画の問題に他ならない．この節では，HTG の視覚化をグラフの自動描画問題として論ずる．

4.1 HTG 描画の方針

HTG を表示する目的は，1 節で述べたように並列化を支援することである．したがって HTG を表示する際にも，並列性がわかるような形で表示されることが望ましい．そこでわれわれは HTG をグラフとして描画する時，次の方針で描画する．

1. 制御フローアークを示すエッジはグラフを表示する画面に対して必ず下向にする．すなわち，画面の上下方向 (縦方向) はプログラムの流れを示す．
2. グラフを表示する画面に対して左右方向 (横方向) は並列性を示す．同じフェーズツリー内で横に並んだノードは並列に実行可能であることを意味する．
3. ユーザはメタノードを展開したりフェーズツリーを併合したりして任意の階層を表示することができる．われわれの目的ではユーザは階層構造にあるノード間の関係を見ることが多いと考えられる．
4. エッジは，制御フローアークを示すエッジ．制御依存アークを示すエッジ，データ依存アークを示すエッジの三種類を表示する．

HTG は，階層を使ってタスクグラフをわかりやすくしていることが特徴なので，グラフ表示の際にもいかにして階層をわかりやすく表示するかが問題となる．次節で階層の表示の仕方について述べる．

4.2 階層の表示法

これまでグラフ自動描画の研究で対象とされてきたのは，木，有向グラフ，平面グラフ，無向グラフといったノード間の隣接関係だけを表したグラフがほとんどであった [13]．これに対し，ノード間の隣接関係と包含関係の両方を表すグラフを複合グラフという．HTG は，依存関係が隣接関係に，階層関係が包含関係にあたると考えられる．

複合グラフの自動描画の問題に関してはまだあまり多くの研究はない．複合グラフのノード間の包含関係を図上でもノード同士の包含関係として視覚化する (この方法を以降ではノード埋め込み方式と呼ぶことにする) ための一般的なアルゴリズムが杉山と三末によって提案されている [9]．このアルゴリズムを使うと一般的にバランスが良い複合グラフを描画することができる．

しかし，われわれは HTG を表示する目的にはこの一般的なアルゴリズムは次の点で不向きであると考えている．

1. グラフが縦方向に伸び，画面の横方向を有効に使えない．HTG はプログラムのタスクの流れを示したものであることから，一般に横よりも縦に長く伸びるという特徴がある．
2. ノードの包含関係が何重にもなると，どのノードがどれくらい深い階層にあるのかが直観的に

わかりにくい．並列化支援の目的にはループのネストにあたる階層関係をわかりやすく見せることが重要であると考えられる．

そこで，われわれは階層関係を，埋め込み方式ではなく，横に表示していく方法を提案する．これを展開方式と呼ぶことにする．これにより画面の横方向をより有効に使うことができると考えられる．単純な展開の場合を図5に示す．ここでグラフのノードのうち円はリアルノードを示し，四角はメタノードを示している．図6, 7にはそれぞれ縦に並んだ二つのメタノードを展開した場合，横に並んだ二つのメタノードを展開した場合を示す．

4.3 考察

前節で提案したHTGの表示方法は埋め込み方式に比べてみやすいのであろうか．同じプログラムのHTGを埋め込み方式で表示した図8と，展開方式で表示した図9を比較してみると，複合ノードを横に展開したことで，われわれの方式の方がノードの大きさを大きく表示できることがわかる．階層関係も階層を示すための補助エッジ(図では点線で表されている)を引くことによって明示することができる．

しかし，実用的なプログラムを視覚化すると，展開方式を使っても全体がわかるような表示を行なうのは難しい．図9が表示しているソースプログラムは70行程度のものである．実用的にはもっと大きなソースプログラムを視覚化することが求められるであろう．実際には注目ノード以外のノードは全部併合された形で表示するという使い方が多くなると予想される．また，並列性を考える上で重要なデータ依存関係を主とした表示方法が必要とされるであろう．これは今後の課題である．

5 関連研究

プログラムの視覚化を行なおうとする研究はいろいろなされている．特に人間にとって，これまでの逐次処理に比べて並列処理を扱うのが難しいことから，並列プログラムに関係した視覚化の研究が多い．ここでは，プログラムの視覚化のうちわれわれの研究に関連の深い並列処理に関係した視覚化やソースコードレベルでの視覚化に関する研究を紹介し，われわれの研究との関連を述べる．

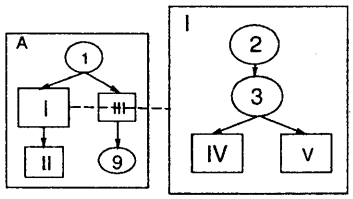

図5: メタノードの中の構造を横方向に展開することによって画面の横方向を有効に使う．左側のIの中身のCFGを表示したフェーズツリーが右側のIである．点線は右側のIが左側のIを展開したものであることを示す．

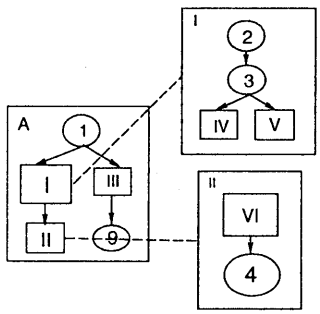

図6: 同じフェーズツリーの中にあることなるメタノードを開いた場合．ここでは，IとIIの二つのメタノードを展開している．二つのメタノードが縦にならんでいた場合それを展開したフェーズツリーも縦に並ぶ．

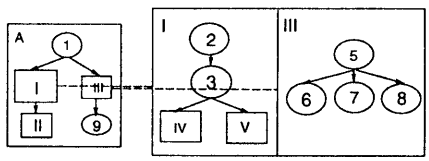

図7: 同じフェーズツリーの中にあることなるメタノードを開いた場合．ここでは，IとIIIの二つのメタノードを展開している．二つのメタノードが横にならんでいた場合それを展開したフェーズツリーも横に並ぶ．

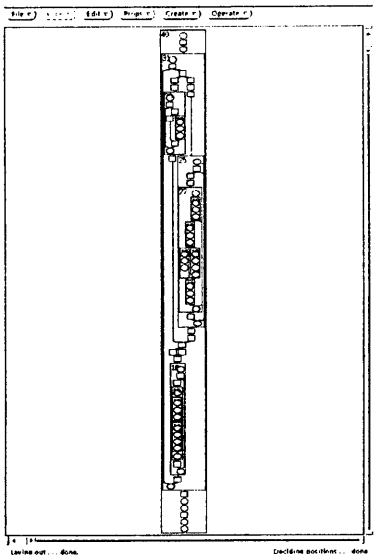

図 8: 埋め込み方式で表示した HTG. (杉山と三末の D-ABDUCTOR による)

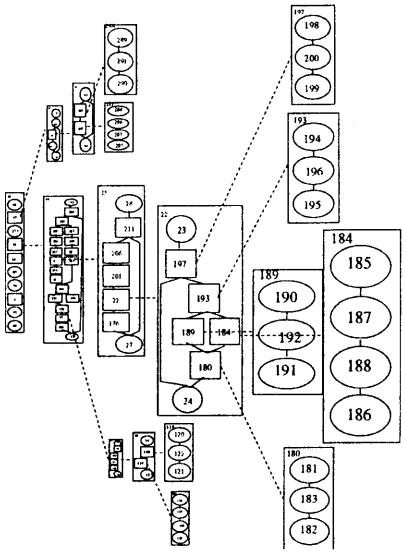

図 9: 展開方式で表示した HTG

並列処理に関する視覚化において，もっとも盛んに行なわれているのが，並列／並行プログラムのデバッグを目的とした実行過程の視覚化である．これに関しては [15] にサーベイがある．われわれがプログラムという静的な情報を扱っているのに対し，これらの研究は実行結果という動的な情報を扱っているという点で異なっている．

リエンジニアリングの分野でも，既存のプログラムを理解するためにソースコードを視覚化する研究が行なわれている [6]．われわれの研究も既存のプログラムの構造の理解が求められるという点でリエンジニアリングの一種であると位置付けることができる．しかし，一般的にこれまでのリエンジニアリングで表示のために扱われるグラフは並列化のためのタスクグラフに比べて簡単であることが多い．

プログラムの理解を深めるため，またはデバッグのためにプログラムの制御やデータの流れを視覚化する研究も行なわれている．HyperDEBU[14] は論理言語用のデバッガでデータフローや制御フローを表示できる．笠原ら [11] は並行プログラムの依存関係をグラフの形で表示する．しかし，これらの研究は単にフローを表示することのみを目的としており，われわれのシステムが目指している複数のビューによる表示やユーザとのインタラクションに関する機能を持っていない．

並列化を支援するための視覚化の研究としては，Novack らの VISTA や Dow らの研究がある．VISTA[7] はわれわれと同じように HTG を基本とした視覚化システムである．ただわれわれのシステムとは違って命令レベルでの並列化を扱っている．したがってソースコードとの対応などの機能はない．一方 Dow らのシステム [3] はわれわれと同じくソースレベルでの並列化を支援する．このシステムは，複数の view を用意し，undo ができるという特徴を持つ．しかし，ここでのグラフ表現は大規模なプログラムの表示には向かない．

6 おわりに

本稿では，並列化支援視覚化システム NaraView の全体構想と HTG のタスクグラフ表示について述べた．今後は今回述べた NaraView を実際に実現し，さらにいろいろな機能を付加して実際に並列化支援に効果的かどうかを確かめる予定である．

追加する機能としては次のことを考えている．

- より多くの種類のビューの提供．例えば，ルー

ブ内での配列に対する参照パターンを示すイタレーションスペースの表示など．そのためにはHTG 以外の中間表現を使うことが必要かも知れない．

- インタラクティブにユーザが情報を与えることによる並列化．ソースコードやグラフ上での操作によって情報を与えることを考える．

参考文献

[1] Aho, A. V. Sethi, R. and Ulman, J. D. "Compilers: Principles, Techniques and Tools", Addison-Wesley, Reading, Massachusetts, 1986.

[2] Barry, R.J. and Evripidon, P. "Extracting parallelism in Fortran by translation to a single assignment intermediate form", Proceedings Eighth International Parallel Processing Symposium, pp.329-34, 1994.

[3] Dow, C.-R., Chang, S.-K.and Soffa, M.I. "A visualization system for parallelizing programs", Proceedings. Supercomputing '92. pp.194-203, 1992.

[4] Girkar, M.B. and Polychronopoulus, C. D. "The Hierarchical Task Graph as a Universal Intermediate Representation", International Journal of Parallel Programming, Vol.22, No.5, pp.519-551, 1994.

[5] Li, Z., Yew, P.-C. and Zhu, C.-Q. "An efficient data dependence analysis for parallelizing compilers", IEEE Transactions on Parallel Distributed Systems, Vol.1, No.1, pp.26-34, Jan. 1990.

[6] Linos, P. K., Aubet, P., Dumas, L., Helleboid, Y., Lejeune, P. and Tulula, P. "Visualizing Programs Dependencies: An Experimental Study", Software – Practice and Experimence, Vol.24, No.4, pp.387-403, 1994.

[7] Novack, S. and Nicolau, A. "VISTA: the visual interface for scheduling transformations and analysis", Languages and Compilers for Parallel Computing. 6th International Workshop Proceedings, pp.449-460, 1994.

[8] 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.

[9] Sugiyama, K. and Misue, K. "Visualization of Structural Information: Automatic Drawing of Compound Digraphs", IEEE Transaction on Systems, Man and Cybernetics, Vol.21, No.4, pp.876-892, July/August 1991.

[10] 伊藤毅志，大西昇，杉江昇 "作図過程を伴う幾何の問題解決認知モデルの提案", 電子情報通信学会論文誌 D-II Vol.J75-D-II No.10, pp.1701-1712, 1992.

[11] 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.

[12] 笹倉万里子，中西恒夫，城和貴，荒木啓二郎 "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 –", 情報処理学会ソフトウェア工学研究会資料 94-SE-99 pp.17-24, 1994.

[13] 杉山公造 "グラフ自動描画法とその応用 – ビジュアルヒューマンインタフェース–", 計測自動制御学会，1993.

[14] 舘村純一，小池汎平，田中英彦 "マルチウィンドウデバッガ HyperDEBU における細粒度高並列プログラムの実行のデータフローの視覚化", 情報処理学会論文誌 Vol.34, No.4, pp.580-594, April 1993.

[15] 長谷川隆三，越村三幸 "並列プログラムおよび性能デバッギングのための視覚化"，コンピュータソフトウェア Vol.12, No.4, pp.32-44, July 1995.

[16] 本多弘樹 "並列処理のためのシステムソフトウェア特集・自動並列化コンパイラ", 情報処理 vol.34, no.9, pp.1150-1157 Sep. 1993.