投機的実行を応用したインタラクティブ並列プログラム

舘村 純一
tatemura@iis.u-tokyo.ac.jp
東京大学生産技術研究所
〒106 東京都港区六本木 7-22-1

投機的並列計算は、余剰プロセッサ資源を用いて要求確定前の計算を見込みで並列実行することで、速度の向上をめざすという考え方である。本研究では、インタラクティブなプログラムに投機的処理を導入して多数のプロセッサを有効に活用し、インタラクションの応答性能の向上だけでなく一定時間内に可能なサービスの内容の充実をはかる。本稿では、投機的並列実行のユーザ・インタフェースへの応用を提案し、研究の構想を概説する。次に、並列論理型言語 KL1 による投機的並列性の記述について述べ、大規模並行プログラムのインタラクティブ可視化ツールをアプリケーション例として検討し、実現のための課題を考察する。

Speculative Parallel Execution of Interactive Programs

Junichi TATEMURA
Institute of Industrial Science, University of Tokyo
7-22-1 Roppongi, Minato-ku, Tokyo, 106 JAPAN

We propose a parallel programming system based on speculative execution for interactive programs. The term "speculative execution" refers to the execution of a parallel program some portions of which will not contribute to the final outcome of a computation. Speculative parallel execution of an interactive program makes effective use of intervals between human-computer interactions, and enhances quality of the service for users as well as rapidity of responses. In this paper, the overview of this programming technique is described. We utilize a parallel logic programming language KL1 to express speculative parallelism of an interactive program. As an case study, we discuss introducing speculative execution to an interactive visualization tool of concurrent programs.

1 はじめに

メモリ容量の増大やプロセッサの速度向上によって、計算機の使われ方は質的に変化してきた。現在のワークステーションなどでは、その計算資源の多くがユーザ・インタフェースに用いられており、人間の知的活動を支援するメディアとしてのコンピュータが注目されている。一方、並列計算機が現在実用化されつつあるが、その利用形態はまだ確定しておらず、利用技術の研究が並列計算機の普及にとって不可欠となっている。また、これらの並列機を含む計算機群が高速ネットワークによって連結される分散環境も豊富な並列計算資源となりうる。人間の数よりもプロセッサの数の方がはるかに多いという環境では、今までには考えられなかったことにもプロセッサを活用できる可能性があり、計算機の新しい利用技術が期待される。

本研究は、知的活動を支援するインタラクティブ・メディアとしての計算機への並列・分散処理の活用を目指し、「投機的並列計算」と呼ばれる概念を導入して、並列計算資源を有効に活用したユーザ・インタフェースを構築することを目的とする。

本稿では、投機的並列実行のユーザ・インタフェースへの応用を提案し、研究の構想を概説する。次に、並列論理型言語 KL1 による投機的並列性の記述について述べ、大規模並行プログラムの可視化ツールをアプリケーション例として検討し、実現のための課題を考察する。

2 投機的並列計算

投機的並列計算とは、近年の並列処理研究の中で生まれてきた概念であり、プロセッサ資源に余裕のある時に、将来利用されると思われる計算をその要求が確定する前に行なっておくというものである。

従来の投機的並列計算の研究対象の中心は条件分岐を含んだ逐次プログラムの並列化であり、命令レベルの細粒度並列処理などを対象としてきた [13]。関数型言語の分野においては、分岐や引数データを予測することによって、関数を投機的に並列実行する手法が考案されている [7] [8]。また、論理型言語の OR 並列処理も投機的な計算として位置付けることができる [17]。投機的計算を積極的にアルゴリズムに導入した例としては、並列イベントシミュレーションにタイムワープ機構を導入した研究例が報告されている [18]。この機構は、並行プロセスがメッセージの到着順を見込んで計算を続行し、到着順序の矛盾が起きた場合は処理をやり直すというものである。

ヒューマン・インタラクションを中心とするプログラムはユーザの入力の逐次性のため、従来の並列化の対象にはされてこなかった。しかし、ヒューマン・コンピュータ・インタラクションには複数のイベントや多数のデータが関わっているので、本質的にはイベント並行性やデータ並列性が存在すると考えられる。本研究では、以下にあげる目的のために上記の投機処理の考え方を応用し、インタラクティブなプログラムのための並列ソフトウェア・アーキテクチャの構築を目標とする。

- **多数のプロセッサの有効利用:** ネットワークで結合されたワークステーション群や、稼働率がピーク時以外の並列計算機などの余剰計算資源を活用する。
- **インタラクティブ・サービスの充実:** プログラム並列化による一定処理の高速実行ではなく、一定時間内の処理量の増大によりサービスの質を向上させるという発想で、並列処理応用を考える。

3 インタラクティブプログラムの投機的並列処理

3.1 期待される効果

投機的並列処理をユーザ・インタフェースに導入することにより、以下のような効果が期待できる。

1. **インタラクティビティ(応答性能) の向上：** 要求を受けてからでは時間のかかる処理のターンアラウンドの短縮が可能となる。
2. **提供する情報の質の向上：** 制限時間までにかけられる計算時間の増加に応じて、データの精度・最適化の度合などの向上をはかる。
3. **先行実行結果のフィードバック：** エラーの事前回避や、ユーザへの提案・予報をアクティブに行なう。
4. **信頼性の向上 ：** 複数の可能性に対応した計算の多重化によって例外的処理への対応を充実させる。

現在、ユーザの意図・希望を推論してアクティブにサービスを行なう知的エージェント技術の研究が行わ

れている [1][6]。このようなエージェントによるアクティブな処理は投機的並列性を本質的に持つので、投機的並列計算を導入すれば (余剰) 計算資源を有効活用した多様なサービスの実現が期待される。例としては、以下のものが考えられる。

- 情報検索・ナビゲーション (要求情報の先行処理、情報リンクの動的構築、アクティブな情報提供、ユーザの観点に基づくデータの集計)
- CAD・発想支援など (ユーザが逐次与える制約条件を満たすような解の探索・提示)
- プログラミング支援 (記述途中の不完全情報の解析によるエラー検出、最適化、デバッグ時の知的支援)
- 知的メニュー/ヘルプ・システム (希望事項リストの予測作成とその先行実行)

3.2 アプリケーションの適性

あるインタラクティブ・アプリケーションに関しての投機処理導入の有効性を判断するには、以下の点を考慮する必要がある。

インタラクションあたりの計算量 要求確定後に十分短時間で計算結果が得られるものは先行処理する必要はない。ある程度の計算量があるタスクが投機実行の対象になる。

全体の計算量・メモリ消費量 利用できる時間とメモリの範囲内であらかじめ全て計算できるものであるなら、インタラクティブに計算する必要はない。通常は要求を得てから計算される処理が投機実行の対象となる。

投機効率 投機タスクの結果の活用効率が低すぎると効果は少ない。総あたり的な単純な予測の組合せでは、可能性の場合の数が爆発してしまうので効果が期待できない。投機実行結果が実際に活用される確率をある程度判断できることが望ましい。

4 並列論理型言語 KL1 による投機処理の記述

4.1 並列論理型言語 KL1

本研究では、インタラクティブプログラムの投機並列処理の記述に並列論理型言語 KL1 を用いる。KL1 は、並列論理型言語 GHC をもとに新世代コンピュータ技術開発機構により設計・実装された言語である。一般の並列言語と比較した場合、論理型言語の「ゴール」という比較的細粒度の処理単位による並列実行の記述と、単一代入変数によるデータフロー的な同期・通信処理の記述が特徴である。

本研究では、KL1 の処理系である KLIC を用いる。KLIC は、KL1 から C へのコンパイラとその実行環境からなるポータブルな処理系である [19]。現在は逐次実行版に加えて、PVM を用いた分散版 KLIC[22] が公開されている。

4.2 言語機能

KLIC では、ゴール GOAL が実行されるノード (ノード番号 NODE) を以下のように指示できる。

```
GOAL@node(NODE)
```

このように明示的に指示しない場合は、ゴールは全てローカル・ノードで実行される。

本研究では、ゴール単位の細粒度並列性による並列実行を目指すのではなく、ノード間のプロセスの動的配置とノード内の並行スレッド実行の柔軟な記述を目的として KLIC の機能を用いる。以下では、ノード内の並行スレッドの投機実行を制御するために利用できる KLIC の言語機能について述べる。

定義節の優先的選択 一つのゴールの実行に対して二つ以上の節が適用可能な場合、alternatively より前の定義節が優先して選択される。

```
p(abort, N, R) :- R = result(N).
alternatively.
p(S, N, R) :- N1 := N + 1, p(S, N1, R).
```

上の例では、第 1 引数が未定義状態の間は再帰実行を繰り返し、第 1 引数が abort という値に束縛されると、一番目の節が選択されて第 3 引数に値を束縛する。

ゴールの優先度つき実行　ゴールには実行優先度が指定でき、大きな優先度を持つゴールが優先して実行される。以下のように、`priority` によって絶対的な優先度の値を指定する方法と、`lower_priority` によって親ゴールに対する相対値を指定する方法がある。

```
GOAL@priority(AbsPrio)
GOAL@lower_priority(RelPrio)
```

ただし、優先度が厳密に反映されるかどうかは実装に依存する。分散版並列処理系では、ノード間にわたったプライオリティの違いは実行には反映されない。

4.3 記述例

ここでは、インタラクティブな処理の投機的並列実行の単純な例を、上記の言語機能を用いて記述してみる。記述例としては、対戦型ゲーム (reversi など) をとりあげる。このゲームは計算機と人間が交互に手を選択して進行していく。計算機は、人間の番の間に人間の可能な手についてそれぞれ次の計算機側の手を並列に思考する。思考部分は、N 手先の各局面を適当な評価関数により評価した結果から打つ手を決定する。計算時間があるうちは、この先読み段数 N を順次深くしていく。人間が手を打った時には該当する探索の一つだけが選ばれ、それ以外の実行は無効になる。

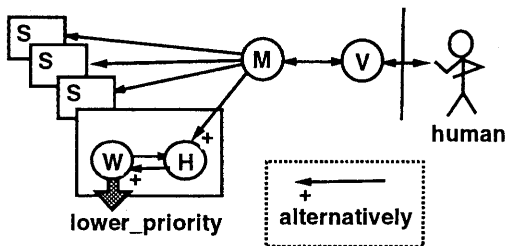

図 1: 対戦型ゲームの投機並列化

プログラムの実行の様子は図 1のように表される。`V` は画面表示・ユーザイベント処理部分、`M` はゲームの思考部分・投機的計算管理部分である。`S` は可能性のある各次局面について先行して思考を行なうプロセスである。`S` の内部はイベント処理部分 `H` と計算部分 `W` の２つのスレッドからなる。`W` は先読み段数 N を順次深くしていきながら繰り返し計算を行なう。N を大きくする毎に (1)`lower_priority` により自分の優先度を低くする、(2) 中間的な評価結果を `H` に報告する、(3)`H` からのメッセージがあればこれに対応する。この (3) の処理を表現するのに `alternatively` を用いる。`H` は、`M` からの要求があれば、現在求まっている評価結果を返す。`H` を `W` と分離するのは、`M` からのメッセージ処理を最優先で行なうためである。`alternatively` は一つのプロセス内でのメッセージの優先的選択を指示するだけなので、実行優先度の低い `W` にメッセージ処理を担当させると、`W` がスケジュールされるまで処理が遅れる可能性がある。

4.4 記述上の課題点

前述の記述例に基づき、Tcl/Tk インタフェースを用いた reversi ゲームを分散版 KLIC 上に作成し、KLIC の言語機能で投機的処理を自然に記述できることを確認した。しかし、実際的なインタラクティブ・プログラムに適用するには以下にあげるいくつかの課題が残った。

KLIC の言語機能のみでは低優先度のゴールプールに入っているゴールを操作できない。そのため、システム全体の実行優先度が低くなってそのゴールがスケジュールされて初めてそのゴールがイベントに対応できる。高優先度のメッセージへの対応は高優先度ゴールに切り分けることで可能だが、不必要になった低優先度ゴールのメモリ資源の回収の問題が残る。

reversi プログラムの実装では、各投機的思考プロセスを各ノードに分配したが、各プロセスの計算量は均質ではなく、また、思考プロセス自体の並列実行も本来行ないうるので、効率的な実行には実行時にノード間の負荷分散が必要となる。このためには、各ノードの負荷を何らかの形で知る必要がある。

また、局面を先読みする場合に、先読み段数に応じて必要なメモリの量が増大する。このため無制限に投機的実行を行なうとメモリ容量を使い切ってしまう危険がある。現在利用可能な計算資源を意識しながら投機的実行を行なう機能が必要である。

さらに、実際的なプログラムでは投機タスクの構成やそのスケジューリング戦略はより複雑になると思われる。問題対象の知識に基づいたスケジューリングの指定をいかに行なうかが課題として残る。

以上のように、投機タスクを priority と alternatively を用いたプロセス (ゴール) として単純に記述するだけでは効率的な投機処理の実行管理が困難と思われる。KLIC の基本機能に加えて、投機処理のための管理機構の部分を別にライブラリとして記述し、アプリケーションプログラムのためのインタフェースを

用意する必要がある。

5 ケース・スタディ — 並行プログラムの可視化ツール

上述した投機処理の記述手法・プログラミング手法・実装手法の具体的な設計を進めるには、実際的なアプリケーションに基づいて検討していく必要がある。本章では、並行プログラムの実行の可視化ツールを例にとって、投機的処理の導入法とその効果について検討する。

5.1 並行プログラムの実行可視化

並行プログラムは、実行の流れが複数あるためにその実行の様子が理解しにくい。これを解決するために、デバッグなどにおいて実行可視化技術が重要視されている。[11][20][4] などは並行プログラムの実行を図的に表現してプログラムの並行性の理解を助けることを目指した例である。これらの手法の大きな課題は、並行プロセスが大量になった場合のスケーラビリティである。

大規模データの表示の問題を解決するには、画面上の表示データを削減するなどしてユーザの理解を助ける手法が重要となる。これは情報可視化技術において共通した課題であり、グラフ構造や木構造の表示に魚眼表示 (fisheye)[9]、hyperbolic plane[5]、フラクタル [16] などを応用した研究例がある。いずれの例も、データ構造の表示にユーザの「視点」を導入し、視点部分を詳しく表示しつつその他の領域も概観がつかめるように表示している。これらの方法は、グラフの構造的な情報のみを用いた汎用的な表示手法である。これに対して、プログラム実行の可視化には、どのような動作に着目するかなどのプログラムの意味上での「視点」が重要であり、上記の手法に加えて、データのクラスタリングなど、グラフの意味的な情報に基づいた抽象化が有効と考えられる。

一方、並列論理型言語のビジュアル·デバッガ Hyper-DEBU [10] では、ユーザが何を見たいのかをブレークポイントとして指定することによって、視点に基づいた情報の抽象化を行っている。この機能は、ユーザの着目するプロセス構造や通信の相互関係の可視化を可能とする。しかし、表示の一部の拡大や詳細な実行情報への視点移動は可能だが、可視化の視点を変えるにはブレークポイントを変更して再実行する必要がある。すなわち、ブレークポイントに基づいた実行情報の可視化部分に対してはダイナミックな視点の移動ができない。一般に、プログラムの構造や動作を良く知らなければ望ましい可視化情報を得るような視点指示は難しく、試行錯誤を通じて適切な表示をインタラクティブに得る機能が望まれる。

以上のように、大規模並行プログラムの可視化には、視点の移動に対してダイナミックに適応しながら情報を抽象化する機能が必要である。ここでは、この機能の高度化を目的として投機的並列実行の導入を検討する。

5.2 システムの基本設計

上記の要請をもとに、KL1 プログラムの可視化システムへの投機的処理の応用を検討する。対象とするプログラムはプロセス指向で記述されたものとする。並列論理型言語におけるプロセス指向のプログラミングとはプロセスネットワークを構成して計算を分散的に行なうプログラムの記述法であり、再帰的なゴール実行で並行プロセスを表現し、単一代入変数を共有することで通信路 (ストリーム) を表現する。プロセスネットワークは動的に構成されるので、ソースコードを観察しているだけではその実行の様子を把握することは難しい。そのために実行情報の可視化を行なうが、実行全体の情報は膨大すぎるため、ユーザがその時持っている疑問点、着目点に対応した、実行の (論理的な) スナップショット (の連続) を抽出して可視化する。

プロセスネットワークのある状態を可視化する場合、視点によって各部の抽象化の度合が違ってくる。プロセス一つ一つの動きに注目するのか、プロセス群としてまとめて抽象化するべきか、多くのデータフロー (通信の依存関係) のうちいずれに着目して実行を見せるか、といった要求に対応して可視化を行なう。

図 2-(a) は、プログラムの実行の可視化作業を表したものである。始めに、ユーザはソースコードから得られる静的情報に基づいて可視化のための指示を行う。システムはこの指示に基づいてどこをどのように可視化するかを解析する。必要な動的情報についてはプログラムの再実行により復元し、これをユーザの観点に基づいて抽象化したものを表示する。この表示結果を観察検討して、ユーザは可視化指示を調整する。

この繰り返し作業ののインタラクティブ性向上を目指し、投機処理の導入を検討する。図 2-(b) はその概念図である。

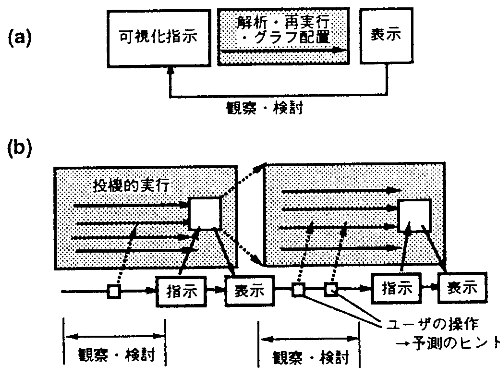

図 2: プログラム可視化作業の投機並列化

5.3 投機処理の導入

5.3.1 投機実行の予測パラメータ

ユーザの可視化指示を予測し、これに基づいて要求される処理を先行実行する。可視化指示は、(1) ソースコードに基づく静的情報に関して、注目するプロセス、通信ストリーム、個々のメッセージ送受信動作、プロセスのグループ化など、(2) 実行情報に関して、実際に生成される個々のプロセス (・グループ) についての同様な指示、が考えられる。

5.3.2 投機タスク

プログラムの部分的再実行 あらかじめ全ての実行履歴を用意するのはメモリ消費量が膨大になるので、ユーザが希望するプログラムの状態を再現するのに必要なだけ再実行を行なう。この要求を予測して投機的に実行履歴を生成する。

プログラム解析 ユーザの視点に応じた可視化を行なうためには、タイプ/モード解析、プログラム・スライシング、データ依存関係による逐次実行部分の抽出などのプログラム解析が有効である。計算資源を投入できればそれだけ詳しい解析が可能となる。ただし、これが実際に投機処理に値するかどうかは、解析のための計算量とその効果を具体的に評価する必要がある。

グラフの最適配置 グラフを視認性よく配置するための自動描画のヒューリスティック・アルゴリズムとして配置問題をエネルギー最小化の問題に帰着させる手法などが研究されている ([21] など)。このような手法では、最適化にかける計算時間に応じてグラフ配置の品質が向上すると考えられるので、投機的実行の対象になりうる。また、表示要求されるグラフ構造を完全に予測できなくても、先行実行により得られた部分解や類似解を初期値に適用することで、品質の良い配置を早く得られる可能性もある。

5.3.3 投機実行のための予測

投機処理の効率向上のためには可能性のある可視化パターンの中から有効と思われるタスクを選択して実行しなければならない。可視化指示の予測には、(1) 部分的に指定された可視化指示、(2) ユーザのブラウズ操作、(3) プログラムの解析結果、などの情報が有効と思われる。この予測に基づき、投機タスクの選択とその優先度の決定を行なう。

5.4 投機的処理の効果

投機的処理導入で期待される最大の効果は可視化指示と結果表示の繰り返しにおけるターンアラウンドの短縮である。この効果はダイナミックな可視化のために重要な要素である。またこれに加えて、(1) アクティブな情報提供 (2) 計算資源に応じた品質の提供、の各効果が期待できる。(1) としては、可視化すべき部分を積極的にユーザに提示したり、バグなどの検知結果を示すことなどが考えられる。(2) にはグラフの配置やプログラム解析の結果が該当する。

5.5 実装上の課題

ここで検討した可視化ツールの例は、4章の記述例に比べると予測パラメータの組合せが複雑である。予想される可視化パターンどうしには、共通する計算部分が含まれる。各可視化パターンにそれぞれ一つの投機タスクを割り当てるのではなく、タスクを分割して共通部分を共有することで計算の効率化がはかれる。このためには、多重環境を導入した実行管理が必要である。可視化指示の組合せが、ひとつの環境 (将来可能な一状態) に対応し、一つの計算は複数の世界に属することになる。一つの環境が選択された時にそれに応じて計算が取捨される。

6 課題と展望

6.1 システムの実装

投機タスクの管理 明示的な投機的分岐プリミティブを導入した並行プロセス言語 Tahiti [3] では、投機処理管理のために「可能世界 (possible worlds)」の概念を導入しており、世界を木構造に階層化してプロセスの複製を減らす管理手法を提案している。また、[12] では、投機プロセスのファイル入出力を管理するために、入れ子構造をなす「世界」をファイル・システムに導入することを構想している。論理型言語の OR 並列処理では、各並列タスクが独自の変数束縛環境を持つ必要があり、各種の実装方法が提案されている [15]。

計算量を削減するためにタスク分割を行なって管理をする場合、一つのタスクの複数の可能世界での共有が生じ、多重環境化が必要となる。[3] ではプロセスが複数の世界に属することを許しているが、プロセス内の条件分岐の処理を対象にしており、知識を用いた柔軟な投機実行を可能にするには不十分である。より一般的な多重環境の表現・管理が必要であろう。

また、管理オーバヘッドを避けるには投機タスクの効率的な分散制御手法が必要である。[14] では、条件分岐の多段先行実行の分散管理を行なうために、各投機タスクが前提条件を情報として持ち、分岐評価結果を受信して各自照合を行なう方式を提案している。前提条件は、条件分岐の結果の列で表されているが、多重環境を表現できるような条件表現を用いることが考えられる。

スケジューリング・負荷分散 負荷分散の方式に関しては、通常の並列処理の場合に加えて優先度の異なるタスク群をいかに負荷分散させるかが課題となる。また、投機的処理は計算資源を有効活用するために行なうので、プロセッサ・ネットワークの負荷、メモリの容量に適応しながら実行しなければならない。

また、ワークステーションクラスタなどの環境ではノード間の通信遅延が大きい。[2] では、この通信遅延を隠蔽するために、メッセージが到着する前にその値を予想して投機的実行を行なっている。ただし、この例では投機的実行とメッセージ受信処理が非同期 (並行) に行なわれていないので、予想が外れた時の損失が大きい。ノード内においては、このような通信遅延の隠蔽を考慮した並行タスクのスケジューリングが望まれる。

6.2 アプリケーション

5章で取り上げた並列プログラムの可視化の例を実装して詳細設計とその評価を行なうことが課題である。それと同時に、その他のインタラクティブ・アプリケーションへの投機処理の導入を考察し、いくつかの具体例について実際にプログラムした上で、アプリケーションの設計技術・プログラミング技法を総合的に検証していく必要がある。

6.3 評価手法

投機的並列処理をインタラクティブなプログラムに導入した場合の性能評価には、システムの並列処理性能と、ユーザ・インタフェースとしてのユーザビリティの両面での評価が必要になる。システムの処理性能に関して従来の並列プログラムの性能評価と異なる点は、実行時間の短縮よりも計算内容や計算結果の質の変化が主眼となることである。このため、速度向上率などとは異なった切口での評価が必要である。

6.4 言語・環境

プログラミング技術、ベンチマーク・アプリケーションの蓄積にともなって、ライブラリの構築や言語機能の設計などが課題となる。

また、投機処理のためのプログラミング環境には、一般の並列プログラミング環境に加えて、優先度記述による投機処理の効率の変化などを観察する機能が重要となる。

7 おわりに

本論文では、投機的並列実行のユーザ・インタフェースへの応用を提案し、大規模並行プログラムの可視化ツールをアプリケーション例として検討した。今後はアプリケーションの実装を行ない、プログラミング手法の構築をはかる予定である。

参考文献

[1] Lisa Dent, Jesus Boticario, et al. A personal lerning apprentice. In *AAAI '92*, pp. 96–103, 1992.

[2] Vasudha Govindan and Mark A. Franklin. Speculative computation: Overcomming com-

munication delays in paralell algorithms. Technical Report wucs-94-03, Washington U. CS, 1994.

[3] Debra S. Jusak, James Hearne, and Hilda Halliday. Implementing speculative pralallelism in possible computational worlds. In *International Conference on Parallel Processing*, pp. II-292–II-296, 1993.

[4] K. M. Kahn. Concurrent constraint programs to parse and animate picutres of concurrent constraint programs. In *FGCS'92*, pp. 943–950, 1992.

[5] John Lamping, et al. A focus+context technique based on hyperbolic geometry for visualizing large hierarchies. In *CHI'95*, pp. 401–408, 1995.

[6] Pattie Maes. Agents that reduce work and information overload. *Communications of the ACM*, Vol. 37, No. 7, pp. 31–40, July 1994.

[7] P. V. R. Murthy and V. Rajaraman. Implementation of speculative parallelism in functional languages. *IEEE Transactions on Parallel and Distributed Systems*, Vol. 5, No. 11, pp. 1197–1205, November 1994.

[8] Randy Osborne. Speculative computation in multilisp. In *Lecture Notes in Computer Science*, No. 441. Springer-Verlag, 1990.

[9] M. Sarkar and M. H. Brown. Graphical fisheye views. *Communications of the ACM*, Vol. 37, No. 12, pp. 73–84, 1994.

[10] 舘村, 小池, 田中. マルチウインドウデバッガ 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. pp. 580–594, 1993.

[11] 高田, 小池. Visualinda: 並列言語 Linda のプログラムの実行状態の 3 次元視覚化. インタラクティブシステムとソフトウェア II(WISS'94), pp. 215–223. 近代科学社, 1994.

[12] 根路銘ほか. 粗粒度投機的並列処理を支援するオペーレーティング・システムの構想. 情報処理学会研究報告 (94-ARC-107), Vol. 94, No. 66, pp. 89–96, 1994.

[13] 山名, 佐藤ほか. 投機的実行の現状と Unlimited Speculative Execution Scheme の提案. 情報処理学会研究報告 (94-ARC-107), Vol. 94, No. 66, pp. 105–112, 1994.

[14] 山名, 佐藤ほか. 並列計算機 EM-4 における多段先行評価の分散制御方式. 並列処理シンポジウム JSPP'94, pp. 257–263, 1994.

[15] 市吉. 論理型言語の並列処理方式. 情報処理, Vol. 32, No. 4, pp. 435–449, April 1991.

[16] 小池, 石井. フラクタルの概念に基づく提示情報量制御手法. 情報処理学会論文誌, Vol. 33, No. 2, pp. pp. 101–109, 1992.

[17] 松田, 鈴鹿, 金田. OR 並列 Prolog におけるプライオリティ制御機構と その応用. 情報処理学会論文誌, Vol. 34, No. 4, pp. 773–781, 1993.

[18] 松本, 瀧. バーチャルタイムによる並列論理シミュレーション. 情報処理学会論文誌, Vol. 33, No. 3, pp. 387–395, 1992.

[19] 仲瀬, 藤瀬, 近山. ポータブル KL1 処理系 KLIC の概要. 情報処理学会第 47 回全国大会, pp. 5-55–56, 1993.

[20] 田中. 並列論理型言語 GHC のビジュアル化の試み. インタラクティブシステムとソフトウェア I(WISS'93), pp. 265–272. 近代科学社, 1993.

[21] 鈴木, 鎌田, 榎本. 単純無向グラフ自動描画アルゴリズム. コンピュータソフトウェア, Vol. 12, No. 4, pp. 45–55, 1995.

[22] 六沢, 仲瀬, 近山. KLIC 処理系の分散メモリ実装方式. 情報処理学会第 49 回全国大会, pp. 5-15–16, 1994.