記 号 処 理 55－2
（1990．6．4）

計算資源指向型並列分散処理システム－Lilac－

安田弘幸、榎本隆昭、前川博俊
ソニー株式会社　総合研究所

計算の高速化や処理機能向上を目的とした並列・分散処理について様々な研究がなされている。我々は、計算における処理単位を資源としてとらえ、資源と資源上での計算とを効率良く処理できるアーキテクチャに基づいた並列分散処理システム Lilac を開発している。Lilac では並列処理の単位や計算の機能を資源として扱い、システム上で記述・表現された並列性とアーキテクチャとの整合性を良くした。これにより、並列処理の記述性を高め実行効率のよい並列処理を実現できる。現在、記号処理分野への応用を目的とした Lilac 実験システムを試作中である。

Lilac: Parallel Distributed System based on Computational Resources

Hiroyuki YASUDA, Takaaki ENOMOTO and Hirotoshi MAEGAWA

Corporate Research Laboratories
Sony Corporation
4-14-1 Asahi-cho
Atsugi, Kanagawa 243
Japan

Parallel distributed processing has been studied in order for improving computational speed and functions. We introduce an architecture based on *resources* recognized as computational processes, which is capable of executing computation on such resources efficiently. We are developing a parallel distributed processing system called Lilac which manipulates decomposed concurrent elements and processing functions as resources using this architecture. The concurrency represented in the system can correspond to computational components effectively so that this architecture can improve capabilities of developing efficient parallel processing system. We implemented an experimental Lilac system applied to the fields of symbolic computation.

1. はじめに

人工知能や自然言語理解など計算機が処理する問題は複雑化し巨大化している。そのため計算機に要求される処理速度や機能はますます大きくなってきている。この状況の中で、計算機の根本的な機能向上は計算機アーキテクチャ開発上での課題のひとつである。

計算機の処理能力の向上の手段として並列処理マシンの開発が行われている。これらのマシンには、共有メモリを持つマルチプロセッサから連想プロセッサ、データフローマシンなど、様々な方式が考えられている [1][2]。しかし、これらは基本的には従来のアーキテクチャを用いたプロセッサを用い、それらの結合方式を工夫することによってシステムを実現しているものが多い。

我々は計算機の能力を高めることを目的とし、並列分散処理に着目し、並列分散処理を効率良く行うことのできる新しいアーキテクチャを持った計算機システム－Lilac－を開発中である。このシステムはすでに我々が提案した計算資源と評価に基づくアーキテクチャ ACRE[3] を用い、高効率な並列分散処理を実現することが可能である。

我々は既存のアプリケーションが多く将来的に並列処理を応用できると思われる記号処理の分野に着目して Lilac システムを試作している。現在、構成要素のひとつである Evaluation Unit (EU) が完成し稼働している。また EU の機能確認とその評価のため、Lisp 1.5[4] と CLOS[5] の処理系をインプリメントした。

本論文では、最初に Lilac における並列処理とアーキテクチャについて述べ、次に試作している Lilac システムとそのプロセッシングユニットについて述べる。

2. 並列分散処理システム－Lilac

Lilac は並列処理における処理単位を計算の資源として捉え、その概念に基づくアーキテクチャを用いた並列分散処理システムである。

2.1. 並列分散処理と計算資源

並列処理には計算の高速化が期待できる。並列処理を行うにはその処理を処理単位に分解する必要がある。これにはハードウエアの計算素子上での並列処理の計算プロセスから、アプリケーション構築における計算コンポーネント、アプリケーション上のデータに至るまで、様々なレベルの分解が考えられる（図1）。Lilac では並列処理を実行するときこれら分解された処理単位を実際の計算機の機構にそれぞれ割り当てる。従来の並列処理では、基本的にその並列性は各階層内で閉じており他の階層の如何にかかわらず並列性を保持するようになっているものが多い。しかしその場合他の階層の処理が不透明になるために、計算モデル上の並列記述での機能とハードウエア上での機能との親和性が得にくくなる。そのためこのような方式で並列処理を実現すれば、予期しないボトルネックが現れたり整合を取るためのスケジューリング機能が必要になるなど、本来の計算実行の妨げとなる要素が現れやすい。

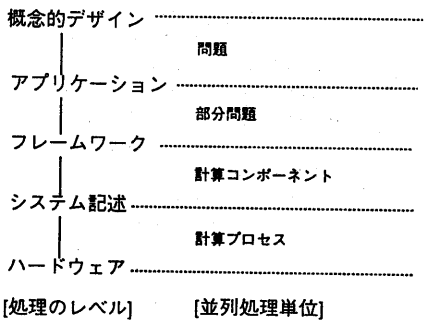


図1. 並列処理の階層

我々はこれら分解された処理単位を計算の資源として考えた。このような計算資源という概念をもとに構築したアーキテクチャ上で計算を行えば、プログラム開発者がアプリケーション上に表現した並列性や分散処理記述をハードウエアの並列動作に反映しやすくなる。各階層での並列処理の単位を資源という形でハードウエア上での計算実行単位に適用させることができる。このような並列処理の実現手法は、システム動作の予測や評価を容易にし、実行効率の良い並列処理システムの構築に有効である。我々はこの考えに基づく計算資源指向型アーキテクチャを構築し、それに基づいた並列分散処理システムLilac を開発している。

2.2. 計算機能の配置と融合

並列処理のため分解した処理単位の計算のための機能を動的に計算機資源に割り当て、それらを有効に利用して計算を行なうことが出来れば、全体の処理の実行効率が上がることが期待できる。我々はこれを計算機能の動的配置と呼ぶことにする。Lilac は、並列処理による高速化をめざすと共に、計算機能の動的配置などを行い、計算機資源を有効に利用して処理効率を向上させることを目的とした。

一般に並列システムにおける並列処理単位の大きさ（粒度と呼ぶ）は、負荷の分散やその処理単位間の通信のオーバヘッドなどを考慮して決定される[6]。細かい粒度の処理では高い並列性を期待することが出来る反面、処理単位間の通信や処理の切り替えによるオーバヘッドが大きくなり、処理効率を低下させる可能性が出てくる。逆に荒い粒度の処理では処理単位間の通信や処理の切り替えによるオーバヘッドはその処理に隠されるが、高い並列性はさほど期待できない場合が多い。

計算機能の動的配置を実現する場合においては、計算における機能と計算機の機構の親和性がその実現性を左右する。親和性が良くなければ、その整合を取るためのオーバヘッドが増加する。並列処理の粒度が細かい場合は、機能の切り分けなどが困難であり、さらに機能の切り替えなどに要するオーバヘッドも増加する。一方粒度が大きければ切り分けの単位が大きくなり、機能の切り替えや配置に要するオーバヘッドも大きくは現れない。また、粒度が荒いために生じる並列性の低下は計算機能を動的に割り当てることによって解消すると期待できる。そこで我々は Lilac において、並列処理と計算機能の動的な分散を実現するために、実装するプロセッサは小数でその機能を高く設定し、並列処理の粒度を大きく取れるようにした。さらに機能配置の自由度が増すように、通信など各プロセッサの機能はすべて同等となるよう設計した。

高機能なプロセッサを複数台持つ Lilac システムにおいては、その処理する機能を適当に配分することによって、異なる機能の処理を融合させたシステムを構築することが可能である。我々はこれを異種機能融合システムと呼ぶ。さらに対象とするモデルを異にするプロセッサを組み合わせることによっても、この機能は実現できる。

2.3. Lilac の基本アーキテクチャ

Lilac におけるプロセッシングユニットは既存技術の制約を受けずに純粋に並列処理の研究を行うため、我々が新たに設計開発した。プロセッシングユニットのアーキテクチャには、我々がすでに提案している計算の評価と資源に基づくアーキテクチャ ACRE を用いた。ACRE は計算の実行時における計算データとメモリ管理やプロセス管理などの資源管理をあわせて資源、計算を実行する過程を評価としてそれぞれ認識し、これらをモジュールとして具現化するアーキテクチャである。

一般に計算モデルを計算機に適用させる場合にはそのアーキテクチャに応じた言語やフレームワークなどの実行モデルを考える。従来はこの実行モデルを実現するにはアーキテクチャ上で種々のシステム記述が必要であり、実行モデルとハードウエアアー

キテクチャとの整合性がよいとは言えない (図2-(a))。ACRE においては実行モデル上での資源と評価という計算の基本要素をアーキテクチャモデュールとして捉えて実現しており、実行モデルとハードウエアアーキテクチャとの整合性を高めている (図2-(b))。

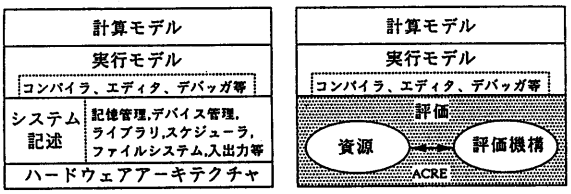


図2. アーキテクチャの比較

資源とは、実行モデル上でのデータの静的表現である。この内部では資源の管理のための処理などが実行されるがそれは評価から見えることはない。評価は、実行モデル上での資源の動的な振る舞いであり、資源に対して変化を与えるものである。

評価機構や資源の機能や領域は適応させるモデルに依存するが、その基本的な構成を図3に示す。

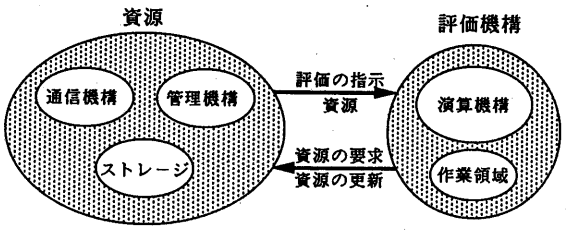


図3. 資源と評価機構の構成

評価機構は、演算処理機構と作業領域とからなる。評価の手順は資源の中にプログラムとして記述する。また評価機構内部にそれを保持することも可能である。

資源は、データおよびその格納場所と記憶管理機構や通信機構、ファイルシステムなどからなる。これらの機構は能動的に資源を操作し環境を構築することができる。評価機構は資源への操作をこれらの機構に対して要求を出すことによって実現する。評価は資源内部のデータ構造や通信プロトコルなどを意識する必要はない。

2.4. ACRE と並列処理

ACRE を用て並列処理を実現する場合、図4で示されるような評価機構と資源の構成が考えられる。(a)は資源を共有する構成、(b)は資源を分散する構成、 (c)はそれぞれ独立している構成である。これらの並列処理においては論理的に、評価機構から見た場合は残りの評価機構とすべての資源はまとめてひとつの資源として捉えることができ、反対に資源からみた場合は他の資源とすべての評価機構はまとめてひとつの評価機構と見なすことができる(図5)。これは並列システムを記述する際に、数多くの機構を意識せずに評価と資源の1対1の関係で捉えることが可能であることを示している。並列処理の個々の処理単位を意識せず ACRE の基本である評価と資源という概念に基づいて効率良く並列処理を記述できることになる。特に (c) の構成の場合は資源と評価が物理的においても1対1であるため、このような捉え方が容易である。評価と資源をもとに記述された ACRE のシステムが並列処理においてもそのまま利用でき、並列処理構築における記述性の向上が期待できる。

3. Lilac の構成とその要素プロセッサ

Lilac は ACRE に基づく Processing Element (PE) を複数台有する並列分散処理システムである。その構成を図6に示す。ACRE での評価機構に相当する部分が Evaluation Unit (EU)、資源に相当する部分が Resources Unit (RU) である。資源は既存のハードウエアを用いて実現するために Resource Manager (RM) と Storage の2つの部分に分け、RM において資源の管理などを行う。外部とのインターフェイスは EU の代わりに Workstation (WS)

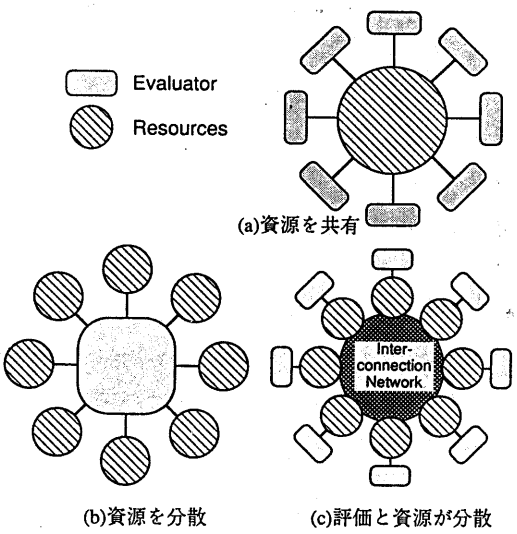


図4. ACREによる並列処理システムの構成

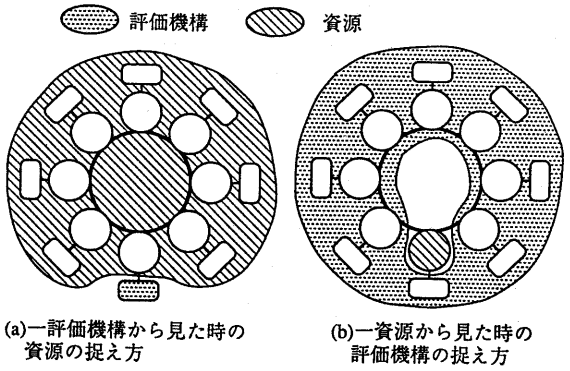


図5. 並列処理システムにおける資源と評価の関係

などを接続して実現する。さらに、記憶領域の拡張のために二次記憶をコントロールする Secondary Storage Controler (SSC) を備えた。

3.1. Lilac 実験システム

我々はこの Lilac の並列処理機能確認のために実験システムを試作している。このシステムでは PE は最大8台（内1台は外部インターフェイス用）実装できる。並列処理の様々な手法の実験を行うため、

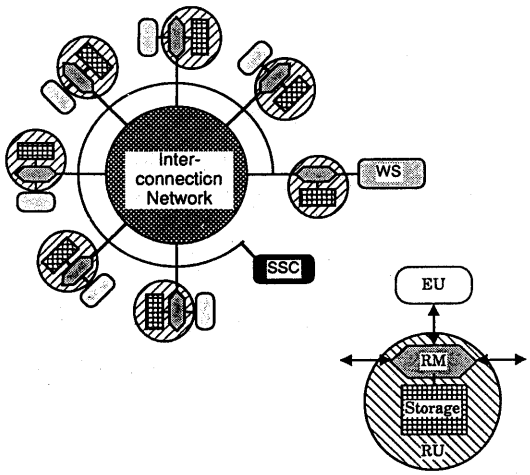


図6. Lilacシステム

多くの既存のアプリケーションが存在しかつ知識処理など将来的に応用が可能と思われる記号処理の分野に着目してアーキテクチャを設計した。実験システムの構成を図 7 に示す。この実験システムは汎用の Workstation (WS) を接続し、その WS 上に Lilac の開発環境を実現した。RU は、EU とほぼ同等の機能を持つ RM と Storage で構成した。この実験システムでは Storage (80bits 幅 64k words) はアクセス効率などを上げるために各 RU に持っているが、そのデータは論理的には共有している。

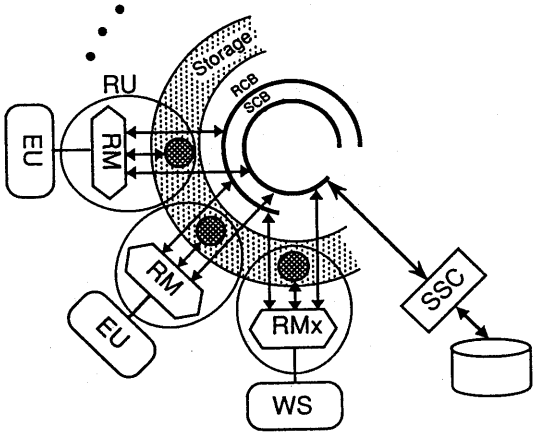
以下にそれぞれの構成要素について述べる。

Evaluation Unit (EU)

ACRE における評価機構に相当する。記号処理を目的にポインタデータなどの高速処理を行えるように設計した。内部は 32bits バス構成でマイクロプログラム制御であり、基本命令サイクルは 125ns である。RM とのインターフェイスを持つ。WCS は 64bits 幅で RM から書換え可能である。この機能は前に述べた計算機能の動的配置、記憶領域拡大などのための仮想記憶に利用する。

Resource Manager (RM)

ACRE における資源の管理などのマネージメン



EU:Evaluation Unit
RU:Resources Unit
RM:Resource Manager
RMx:RM for external systems
SSC:Secondary Storage Controller
RCB:Resource Communication Bus
SCB:Secondary Storage Communication Bus
WS:Workstation

図7. 実験システムの構成

トを行う機構である。RM は EU と同様の基本構成を持つプロセッサで、内部は32bits バス構成でマイクロプログラム制御、基本命令サイクルは 125ns である。WCS は 64bits 幅で WS から書換え可能である。ACRE においての資源間の結合は RM の RCB(後述)インターフェイスを利用する。

Lilac での記憶管理は、すでに我々が提案している SOSO[7][8][9] を用いる。これはポインタで繋がれたデータを管理する手法であり、主記憶と二次記憶からなる記憶構成において、時間的、空間的に処理効率のよい記憶管理を行なうことができる。SOSO の記憶管理は主に RM で行なう。

RMx は外部とのインターフェイスのための特別な RM である。基本的な機能は RM と同じであり、さらに外部とのインターフェイスのための機構を備える。

Secondaly Storage Controler (SSC)

Lilac 全体の二次記憶のサポートを行う。SSC は各 RM とバスを介して通信し、二次記憶へのデータの待避や二次記憶からデータの復帰を行う。SOSO における二次記憶管理は SSC で行なう。

Resource Communication Bus (RCB)

RM 間を結ぶ通信バスであり、60bits 幅である。主に資源間通信に利用する。通信は特定の資源同士や、資源全体を対象とするなどそのパケットの内容によって区別して行われる。各 PE は非同期にこのバスを使用するため、その調停は RMx が行う。バスの使用権の優先順位は時間と共に変化し、各 PE のバス使用権獲得の期待値は均等になる。

Secondary Storage Comunication Bus (SCB)

各 RM と SSC 間の通信バスであり、23bits 幅である。SOSO において RM にある主記憶から情報を SSC にある二次記憶に待避させたりそこから主記憶へ復帰させたりするときなどに利用する。バスの調停は SSC が行なう。二次記憶のアクセスと資源間の通信との競合を避けるために SCB と RCB を独立させた。

3.2. EU のアーキテクチャ

現在 Lilac の要素プロセッサである EU がすでに完成している (図8) 。基本命令サイクルは 125ns である。スクラッチ・パッド・メモリ (SM)とシステム・スタック (SS) は共に 32bits幅で 64k words 実装している。SM は1クロックサイクルでリードとライトを両方実行することができ、高速アクセスを実現している。内部レジスタはこの SM 上に実現した。また、EU 単体でデバッグや機能確認などができるように EU に Storage を実装した。
以下に特徴について述べる。

データとアドレスの区別のない内部バス

内部バスは3本あり、既存の多くのプロセッサにあるようなアドレスバスとデータバスの区別がない。リストデータを扱う際にアドレスとし

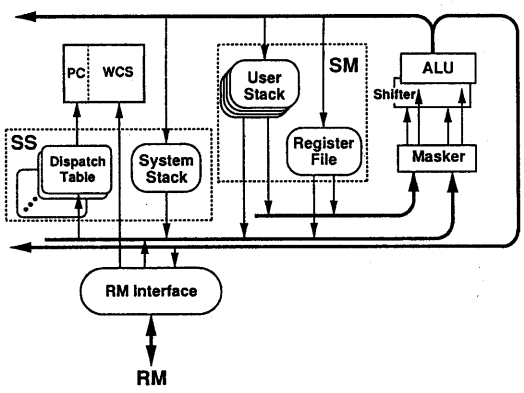


図8. EUの構成

て表現されているポインタを EU 内部でデータとして扱いやすいバス構造とした。

高速ハードウエアスタック

EU 内部の SM には自動増減機構付きポインタレジスタを4本有し、それを利用してデータをアクセスすることにより、ハードウエア・スタックとして利用可能である。さらに、そのポインタレジスタをベースにオフセットをかけてのアクセスも可能であり、ローカル変数や関数呼出しの引数のアクセスなどが高速に実行できる。さらに、SM の領域の一部は汎用のレジスタファイルとして使用できる。これらはポインタレジスタを介してもアクセス可能である。

多方向分岐機能

1命令で多方向に分岐可能な多方向分岐命令を備えている。これは与えられたデータの指示されたフィールドの値によって SS 上にあるテーブルを参照し、その分岐先を決定する。最大256 方向まで設定できる。この機能はタグによる機能別の分岐や各関数やメソッドなどへの分岐などに利用できる。

命令レベルの並列処理

マイクロプログラムにおいて、1命令内に加減算などの演算命令と分岐命令を記述することができる。記述された演算命令の演算の結果（状態フラグ）に従った分岐が実行できる。これを Actual Condition 分岐と呼ぶ。従来のプロセッサにある条件分岐ではひとつ手前、もしくはそれ以前の演算の結果を反映するのが普通であるが、この機能では同一シーケンスで条件判断の演算と分岐が実行でき、処理効率の向上がはかれる。

Writable Control Storage(WCS)

EU の WCS は 64bits 幅で 64kwords 実装している。書き換えは RM が行なう。この機能を用いて Lilac の特徴である計算機能の動的割り付けが可能になる。また、このメモリはデマンドページング方式[10]の仮想記憶化を行っている。仮想記憶の処理は RM が行う。WCS を仮想記憶化することによって、記憶空間が広がり、より大きなプログラムが実行可能になるのに加えて、さらに、仮想空間が実現できることによる記憶空間の有効利用と、マルチプロセス環境でのコンテキスト・スイッチングの実現が容易になる。

診断、統計収集機能

FP には診断用のインターフェイスと統計収集用の機能が実装されている。診断用は各レジスタの状態やバスの状態などを、本体の回路をなるべく通さずに見ることができる。これはハードウエアのデバッグやソフトウエア開発に役立つ。統計収集機能には、設定した命令の実行回数や設定した範囲の実行時間などが測定できる機能があり、システムの評価や改良に利用できる。

3.3. EUの機能確認

EU の機能確認とその評価のため、Lisp 1.5 準拠のインタプリタと CLOS サブセットのインタプリタを EU 上にインプリメントし、ハードウエアの機

能確認を行なった。現段階ではLilac の機能としては EU のみの稼働であり、EU と WS とを疑似的なインターフェイスを作成し接続して稼働させた。プログラムは EU 側にインタプリタ本体、WS側には入出力関係の処理である。CLOS の処理系についてその概要を以下に示す。

- 記述言語　EU側：マイクロアセンブリ言語
  WS側：C言語
- プログラム容量　EU側：約 9000 行
  WS側：約 5500 行
- クラスの継承は単一継承のみ
- メソッド結合は standard のみ

これらの実際のインプリメントにおいては、リストセルのタグによる機能別への分岐やメッソッドへの分岐などに多方向分岐を使用し、さらに、各所の条件分岐には Actual Condition 分岐を使用することにより、プログラムの記述性が向上した。EU の特徴的機能を使用し効率の高い処理の実現が期待できる。

4. まとめ

我々は並列分散処理における処理単位を計算の資源として捉え、その概念に基づくアーキテクチャを用いた並列分散処理システムLilacを開発している。Lilac では、資源としての処理単位とハードウエアアーキテクチャとの整合性が良く、記述した並列処理を実際の計算機構での計算に反映させ易い。従って、システム動作の予測や評価が容易になり、実行効率の良い並列システムの実現がしやすくなる。さらに Lilac では計算の機能を計算機構に動的に割り付けることにより、異なる機能の計算を効果的に処理することができる。Lilac システムの基本アーキテクチャとしては我々がすでに提案しているアーキテクチャ ACRE を用いた。ACRE は計算における資源と評価をアーキテクチャ構築の基本単位としており、Lilac における並列処理の扱いや計算機能の配置を効率良く実現できる。

現在我々は Lilac 実験システムの内、Evaluation Unit の実装を終え、その機能を確認した。また、Resources Unit や Secondary Storage Contoroler などを製作中である。

我々は今後、Lilac 上の並列分散処理システムを構築することにより、 Lilac の並列分散処理機能の確認、負荷分散などのための種々の並列処理手法の評価、異種機能融合システムの実験などを行なう予定である。

謝辞

本研究の遂行にあたっては、福田譲治室長をはじめとする当研究室のメンバーの支援を受けた。特に Lilac のアーキテクチャに関する沢田佳明、實藤隆則両氏との議論は大変有益であった。

本研究の機会を与えて頂いた当社総合研究所の宮岡所長、ならびに総合研究所情報通信研究所の松田所長に感謝する。

【参考文献】

[1] 富田眞治 他: 並列処理マシン, オーム社 (1989)

[2] 高橋義造: 並列処理マシン開発の現状,情報処理,Vol.28, No.1, pp.10-18 (1987).

[3] 前川博俊 他: 計算における評価と資源に基づくアーキテクチャ, 情報処理学会第40回全国大会, 5L-1, pp.1225-1226 (1990).

[4] J. McCarthy, et al.: Lisp 1.5 Programmer's Manual, MIT Press (1962)

[5] Daniel G. Bobrow, et al.: Common Lisp Object System Specification, X3J13 Document 88-002R (1988)

[6] 柴山 潔: 記号処理マシン, 情報処理, Vol.28, No.1, pp.27-46 (1987).

[7] 前川博俊 他: Pointer-Linked Data における仮想記憶管理の一手法, 情報処理学会研究会資料, SYM50-1 (1989).

[8] 前川博俊 他: Linked Data Structures の記憶管理, 情報処理学会第39回全国大会, 1Q-4, pp.1279-1280 (1989).

[9] 前川博俊 他: Linked-Data のその構造に基づく記憶空間の構成, 情報処理学会研究会資料, ARC80-5 (1990).

[10] Harvery M. Deitel: An Introduction to Operationg Systems, Addison-Wesley Publishing Company (1984).