図的データ駆動言語による
高度並列処理方式に関する研究

１９８４年１月

西　川　博　昭

内　容　梗　概

本論文は筆者が大阪大学大学院工学研究科後期課程（電子工学専攻）在学中に寺田研究室において行った研究のうち，図的データ駆動言語による高度並列処理方式に関する研究をまとめたものであり，次の６章をもって構成する。

第１章序論においては，本研究の目的ならびにその工学上の意義，及びこの分野での研究の現状について述べ，本研究で得られた新しい成果について概説している。

第２章においては，高度並列処理方式の実現手法として，従来のハードウェア先行形のシステム開発過程の反省に基づく，言語主導形の実現法について述べている。即ち本研究では，高度並列処理に関して最適な表現形式を採用した言語を，問題向き利用者言語と実ハードウェアの間に位置する中間言語としてまず定式化し，これに副作用を回避できる厳密な実行規則を付与すると共に，最終的にこの言語の実行規則を最も効率よく実現する並列処理機構を模索する手法をとっている。まず高度並列処理方式の満たすべき要件として，１）対象とする問題に内在する並列性を余すことなく活用できる柔軟な機能および負荷分散処理，２）分割損のない効果的な分散制御，並びに３）システムと外界との効率の良い入出力インタフェース，を明確にしている。次に，高度並列処理実現に，従来の文章形記述に代わる図的表記に基づく中間言語およびそれにより表現される履歴依存性を許すデータ駆動原理が有効であることを示している。

第３章においては，副作用の生じない並列処理構造を陽に記述できる図的データ駆動言語（Diagrammatical Data-Driven Language；D^3L）を提案している。まず D^3L に採用した図的表記法であるデータ駆動図式の仕様，並びに D^3L によって記述される階層的なプログラムに副作用が生じない保証を与える検証性について述べている。次に，履歴依存処理の記述のため拡張されたデータ駆動図式についても同様の検証性が保証されることを示すと共に，図式中の履歴の具体的実現例として，副作用を回避できる参照・更新規則を持つタグ付き記憶セルを提案している。

第4章においては，D^3L によって表現されるデータ駆動原理の一実現法について述べている。まずデータ駆動原理による高度並列処理の実現には，D^3L プログラム中に陽に示されるアルゴリズムに基づく並列処理のみならず，入力ストリームに応じたデータ流をシステム内に確保できる実行時の動的な負荷分散が必要であることにふれている。次に，システムの動作をモデルにより解析し，高度並列処理実現のための前提条件として，1）対象とする D^3L プログラムの分割割当，及び2）並列実行されるタスクレベルの設定，の適正化を示している。更に，D^3L プログラムのデータ駆動形実行のための基本機能群を示した後，これらの条件を満たす実行制御の一方式として，D^3L プログラムのブロック構造を反映した階層的なクラスタの繰返し構成をとるシステム上でこれらの基本機能群の負荷ならびに機能分散的実行を統一的に実現できる資源割当動作モードを持つ実行制御方式について述べている。

第5章においては，D^3L による高度並列処理の実現手法を実験的に検討している。まず共通バス形式のマルチプロセッサ構成をとる実験システムの特徴およびその上での D^3L の一実行形式ならびに処理系について簡単にふれている。次に，階層的システムへのプログラムの分割割当および並列実行されるタスクレベルの設定が適正であれば，本方式が高度並列処理を実現しうることを実験結果に基づき示している。更に，対象とする D^3L プログラムの構造に基づき，そのデータ駆動形実行のための各基本機能を階層的なクラスタ構成上へ分割配置するための指針を半定量的に与えている。

第6章結論においては，本研究で得られた結果と残された問題についてまとめている。

関 連 発 表 論 文

(1) 浅田，寺田，白野，渡辺，磯，国本，西川：“Passive memoryless architecture上のデータフロー形実行制御”，信学会電子計算機研究会資料，EC80-49 (1980-12).

(2) 寺田，浅田，西川，国本，磯，矢野，坂口：“ＰＭＡによるデータ駆動形マルチプロセッサシステム”，情処学会マイクロコンピュータ研究会資料，18-2 (1981-09).

(3) 西川，寺田，浅田：“図的表現によるデータ駆動形並列処理記述言語について”，信学会電子計算機研究会資料，EC81-74 (1982-02).

(4) 矢野，西川，浅田，寺田：“データ駆動形マルチプロセッサシステムの一実行制御方式”，信学会電子計算機研究会資料，EC82-37 (1982-07).

(5) Nishikawa, H., Asada, K. and Terada, H.：“A decentralized controlled multi-processor system based on the data-driven scheme”, Proc. 3rd Int'l Conf. on Distributed Computing Systems, pp. 639-644 (Oct. 1982).

(6) 西川，寺田，浅田：“履歴依存性を許すデータ駆動図式”，信学論（D），J66-D, 10, pp. 1169-1176 (1983-10).

(7) 西川，浅田，寺田：“データ駆動形実行制御の一方式とその実験的検討”，信学論（D),(投稿中）.

図的データ駆動言語による高度並列処理方式に関する研究

目　　次

第1章　序　　論

ＶＬＳＩ技術により計算機システム性能を飛躍的に向上させるため，素子動作速度の物理的限界の下で，膨大な素子数を有効に活用できるシステムの構成手法の確立が望まれている[(1)-(8)]。素子技術の制約下で，システムの処理能力を極限まで追求すれば，最終的には対象とする問題向きに専用化された配線論理によるシステムに到達する。これは，いわば論理ゲートの並列動作性を限界まで利用する方式である。現在でも，多くの高性能システムが逐次形のプロセッサの処理速度の制約から，やむなくこの方式を採用している。しかし，ＶＬＳＩの設計・検査限界の到来を予想できる現状では，適切な判断とは言い難い。事実，ＶＬＳＩの製造と設計技術のギャップは拡大しつつあり，現在製造技術の許しうる集積度に近いレベルで設計可能なものは非常に規則性の高いメモリチップのみである。ＶＬＳＩ設計・検査技術の今後の発展を考えあわせても，システムの実現方式は規則的あるいは組織的な構成をとらねばならない。この現状を背景に，シストリックアレイ[(3)]などが盛んに研究されている。しかしながら，更にＶＬＳＩの大量生産による価格の低下および信頼性の向上を考慮すると，ある程度の汎用性を有する方式が望ましい。以上の観点より，このようなシステムの実現には，プログラム制御方式を採用せざるを得ない。この要請にこたえる一手法として，対象となる問題に内在する並列性が許す限り，システム資源の投入に見合う処理能力向上を可能とする高度並列処理方式が注目されている[(1),(2),(5),(7)]。

アレイ／パイプライン処理方式あるいは通常のマルチプロセッサシステムなどに代表される並列処理方式は早くから各所で研究され，さまざまな成果が報告されている。しかし，前者は物理的なデータ構造，ハードウェア構成などを対象としているため，特定の演算に限定的な高能力を発揮するが汎用性に欠ける。又，後者はそのほとんどが在来形の逐次実行形式のプロセッサを基礎としたため，プロセス同期に関するプログラム技法上の困難と実行上のオーバヘッドとを免れないなど種々の問題点が指摘されている。従って，マルチプロセッ

サシステムも，並列性の高いアルゴリズムの適用が可能な問題向きに，専用システムとして構成されることが多かった。ＶＬＳＩ技術の観点[3]からすれば，高度並列処理方式をなるべく斉一なモジュールの繰返し構成で実現することが望ましいと考えられ，均一マシンシステムやトリーシステム[1],[2]などが研究されているが，現在のところ決定的な方式は提案されていない。この原因としては，並列処理方式の適用範囲が限定的であり且つプログラミングに未解決の問題が残されていることが一般に指摘されている。しかし，その適用範囲は，高度並列処理方式の実現手法の確立と共に拡大されると予想すると，高度並列処理実現上の本質的な問題点は，プログラムの記述法およびその実行制御が必ずしも簡単でないことにあると考えられる。高度並列処理の実現には，同時並行的に実行される多数の処理の間の相互干渉，いわゆる副作用がないことが前提条件になる。更に，このような多数の処理の並列実行を効果的に駆動できる分散形の制御原理が必要となる。従来形の単一プロセッサ上の実時間制御プログラムの設計上の困難と複雑さからみれば，多数のプロセスの調停が要求される高度並列処理システムの組織化などは至難とすることは，至極当然の結論と言わざるを得ない。従って，高度並列処理方式の実現には，従来のプログラミングスタイル及び逐次処理プロセッサの概念から脱却した新しい手法を採用しなければならない。

これまでの計算機システムは，高価なハードウェアがまず開発され，プログラミング言語が後を追う形で発展してきた。最近では，これがいわゆるソフトウェア危機を招いた重要な原因の一つであるという認識と共に，ハードウェア技術の格段の進歩により，その相対価格が低下したので，プログラミング言語を機械に従属させた方法が言語体系に歪を与えたことを反省する気運が高まっている。本研究では，高度並列処理実現に際して，ソフトウェア危機の一層の助長を避けるため，従来の計算機システムの研究・開発過程を反省し，まずプログラミングの容易さを主眼として，高度並列処理に関して最適の表現形式を採用した言語を定式化し，これに副作用の回避を保証できる厳密な実行規則を付与すると共に，最終的にこの言語の実行規則を最も効率良く実現する並列処

理機構を探索する手法をとる。従って，本研究において，この条件を満たすプログラムの記述法の確立が最も重要となるが，並列処理記述言語として旧来の逐次処理記述の概念を脱却した，新しい記述法が既にいくつか提案されている。例えば，単一割当言語[9],[10]はプログラム中の各変数の定義を高々一回に制限し，その中に陰に含まれる並列性の検出と検証性の向上を意図している。しかし，この方法は，本質的に逐次記述の概念を伴う文章形の代入文を用いるので，並列処理の了解性に欠ける。又，データフロー言語[11]-[25]は，各処理のデータ従属性を示す図式に基づいた記述形式を採用し，データ駆動の概念による並列性の自然な表現をめざしている。しかし，この言語は記憶の概念を持たないため，履歴依存性を有する処理[26]を直接的には表現できない。しかしながら，この図式は，文章形記述には求められない本質的な利点を持っている。即ち，データ従属性を示す図式は記述形式そのものに，アルゴリズムに内在する並列処理構造（同時並行処理，パイプライニング）を陽に反映している。又，階層的に構造化された記述をとれば，このような図的表現は，大規模なプログラムの記述法としても，優れた了解性を維持できる。そこで本研究では，履歴依存処理を含めた並列処理構造を記述形式に陽に反映し，且つ副作用を生じない階層的な並列処理記述法として，履歴依存性を許すデータ駆動図式[27]-[30]を採用し，この図的言語を物理的並列処理機構から独立で且つ利用者言語と実ハードウェア構造との間に位置する中間言語としてまず定式化する。この中間言語としての位置付けは，現状のソフトウェア危機を背景にした仕様記述の高水準化，並びに上述したような高度並列処理方式における副作用の回避のための厳密なプログラムの記述という背反する要求を考慮したためである。従って本研究では，更にこの中間言語に厳密な高度並列実行規則を与えると共に，その最適な実行形式，即ち高度並列処理方式の実現法の実験的検討を目的としている[29],[31]。

本論文では，これらの要件を満たす中間言語の一つとして，図的データ駆動言語（Diagrammatical Data-Driven Language ; D^3L）[30]を提案すると共に，これにより，1）高度並列処理に必要な同時並行／パイプライン処理を含む並列処理構造が陽に記述できること（了解性），2）履歴依存性を有する処理に

関しても副作用のない実行が保証されること（検証性），３） D^3L の仕様に基づく自然な実行形式を採用すれば問題の構造・規模に適応できる拡張性を持つ実行制御方式が構成可能なこと（階層性），並びにその実験的検討によって，４）この D^3L の一実現法が高度並列処理を実現しうること，５）高度並列処理方式における基本的な機能構成が対象とする D^3L プログラムの幾何学的接続構造より近似的に求まること，などについて考察している[31]。

以下に本論文の構成，並びに新しく得られた成果を述べる。

第２章においては，高度並列処理方式の実現手法として，従来のハードウェア先行形のシステム開発過程の反省に基づく，言語主導形の実現法を述べる。即ち本研究では，高度並列処理に関して最適な表現形式を採用した言語を，問題向き利用者言語と実ハードウェアの間に位置する中間言語としてまず定式化し，これに副作用を回避できる厳密な実行規則を付与すると共に，最終的にこの言語の実行規則を最も効率よく実現する並列処理機構を模索するという手法を採用した背景にふれている。まず高度並列処理方式の満たすべき要件として，１）対象とする問題に内在する並列性を余すことなく活用できる柔軟な機能および負荷分散処理，２）分割損のない効果的な分散制御，並びに３）システムと外界との効率の良い入出力インタフェース，を明確にする。次に，高度並列処理方式の実現において，従来の文章形記述に代わる図的表記に基づく中間言語およびそれにより表現される履歴依存性を許すデータ駆動原理が物理的高度並列処理機構の模索のみならず，問題向き利用者言語あるいは仕様記述過程の定式化に対しても有効であることを示す。

第３章においては，副作用の生じない並列処理構造を陽に記述できる D^3L を提案する。まず D^3L に採用した図的表記法であるデータ駆動図式の仕様を示すと共に，D^3L によって記述される階層的なプログラムに副作用が生じない保証を与える検証性について述べる。次に，履歴依存処理の記述のため拡張されたデータ駆動図式についても同様の検証性が保証されることを示すと共に，図式中の履歴の具体的実現例として，副作用を回避できる参照・更新規則を持つタグ付き記憶セルを提案する。この記憶セルは，履歴依存性を許すデータ駆

動形実行環境における副作用の回避を必要最小限の履歴のコピーによって可能とするものである。

第4章においては，D^3Lによって表現されるデータ駆動原理の一実現法について述べる。まずデータ駆動原理による高度並列処理の実現には，D^3Lプログラムが陽に示すアルゴリズムに基づく並列処理のみならず，入力ストリームに応じたデータ流をシステム内に確保できる実行時の動的な負荷分散が必要であることにふれる。次に，システムの動作をモデルにより解析し，高度並列処理実現のための前提条件として，1）対象とするD^3Lプログラムの分割割当，及び，2）並列実行されるタスクレベルの設定，の適正化を示す。更に，D^3Lプログラムのデータ駆動形実行のための基本機能群を示した後，これらの条件を満たす実行制御の一方式として，D^3Lプログラムのブロック構造を反映した階層的なクラスタの繰返し構成をとるシステム上で，これらの基本機能群の負荷ならびに機能分散的実行を統一的に実現できる資源割当動作モードを持つ実行制御方式について述べる。この方式は，ある特定の問題向きの専用システムを構成する際の基本機能構成などの設計指針を提供するものである。

第5章においては，D^3Lによる高度並列処理の実現手法を実験的に検討する。まず共通バス形式のマルチプロセッサ構成をとる実験システムの特徴およびその上でのD^3Lの一実行形式ならびに処理系について簡単にふれる。次に階層的システムへのプログラムの分割割当および並列実行されるタスクレベルの設定が適正であれば，本方式が高度並列処理を実現しうることを実験結果に基づいて示す。更に，高度並列処理の実現にはデータの追越しのない実行環境が必要であることを述べると共に，対象とするD^3Lプログラム構造に基づきそのデータ駆動形実行のための各基本機能を階層的なクラスタ構成上へ分割配置するための指針を半定量的に与える。

第２章　高度並列処理方式と図的データ駆動言語：D^3L

2.1　緒　　言

本章では，従来のハードウェア先行形の計算機システムの研究・開発過程の反省に基づき，ＶＬＳＩ技術を効果的に活用できる高度並列処理方式を実現するための一手法として，次のような言語主導形の開発手法について述べる。即ち，高度並列処理に関して最適な表現形式を採用した言語を，問題向き利用者言語と実ハードウェアの間に位置する中間言語としてまず定式化し，これに副作用の生じない厳密な実行規則を付与すると共に，最終的にこの言語の実行規則を最も効率よく実現する並列処理機構を模索する。これは既に述べたように，ＶＬＳＩの微細化技術の限界に到達した状況において，汎用性があり且つ組織的な構成を持つ処理方式でない限り，飛躍的に増大した素子数の活用は不可能であると言う立場から，プログラム制御方式を高度並列処理の実現法として採用し，このためには，従来のプログラミングスタイル及び逐次プロセッサの概念から脱却した新しい手法を採用しなければならないと考えるためである。

以下本章ではまず，並列処理による性能向上の常套的手段として従来から採用されている，アレイ／パイプライン処理方式，マルチプロセッサシステムについて概観し，これらの手法が物理的なデータ構造，ハードウェア構成あるいは逐次プロセッサの概念から出発したものであり，高度並列処理の実現手法として，そのまま発展しうるものでないことを明らかにする。同時に，高度並列処理方式の満たすべき要件として，１）対象とする問題に内在する並列性を最大限に活用できる柔軟な機能および負荷分散処理，２）分割損のない効果的な分散制御，並びに，３）システムと外界との効率の良い入出力インタフェース，を明確にする。次に，高度並列処理方式の実現において，簡明で且つ厳密な解釈を付与された図的中間言語とそれによって陽に表現される並列処理構造を反映したデータ駆動原理が有効であることを示す。又，これらが現状のソフトウェア危機への対処，利用者言語あるいは仕様記述過程の定式化に対しても有望であることも併せて示している。

2．2　高度並列処理の実現手法

計算機システムは，具体的には以下の項目をコストとバランスさせながら向上させることを目的に発達してきた。

・処理能力／応答時間

・機能（処理，記憶，通信，入出力）

・ＲＡＳＩＳ，但し，Ｒ：Reliability（信頼性），
Ａ：Availability（可用性），
Ｓ：Serviceability（保守性），
Ｉ：Integrity（保全性），
Ｓ：Security（機密性）．

・易用性（ユーザインタフェース）

ここで，上述の項目のうち特に，システムの処理能力／応答時間の向上およびその価格性能比の改善手法として注目されてきた並列処理の概念は既に100年以上も前にBabbageによって提案されており，決して新しいものではない。

【定義２．１】 並列処理とは，処理対象とする問題，即ち一つのジョブを幾つかの独立した処理単位（プロセス）[32]に分割し，これらを複数の処理モジュールを用いて処理することである。但し，プロセス間には，通常データの授受を含む依存関係が存在する[33]。

現在までに多数の並列処理方式が研究され，さまざまな成果が報告されている。まず処理対象とするデータ構造に着目した並列処理方式として，アレイ計算機が挙げられる。この典型的な応用例が天気予報である。気象現象は地球上の大気を3次元のメッシュとしてこの上での偏微分方程式としてモデル化できる。このメッシュの細かさ，メッシュポイント上での変数の種類（例えば，気温，風向その他）等によってモデルの複雑さが決定される。典型的なモデルにおいては，1メッシュポイント当たり約1000演算が必要であるとされている。このモデルでメッシュの規模を，垂直方向に12レベル，水平方向では緯度および経度各2度ごとに1ポイントとし，実時間の100倍の速度で処理するためには約80MIPSの計算機が必要となる。即ち，従来形の逐次処理計算機の処理能

力では不充分であり，アレイの独立した各要素ごとの演算を同時実行する並列処理によって速度を向上させなければならない。このような状況の下では，処理能力の向上が優先し経済性がある程度悪くても正当化され得る。このようなアレイ処理方式の具体的な実現例としては，ILLIAC Ⅳ[34]などの専用システムが挙げられる。ILLIAC Ⅳは，図2.1に示すように対象とするアレイ構造を直接的に反映した構成をとる。即ち，制御装置（Control unit；CU）の下に64個の演算装置（Processing element；PE）が接続される。各PEには2048語の固有のメモリ（PE Memory；PEM）が付与されている。例えば，64個の要素を持つ二つの行列A，Bの加算では，A，Bのi番目の要素Ai，Biがi番目のPEのPEMiに格納されていれば，行列としてのA＋Bは，す

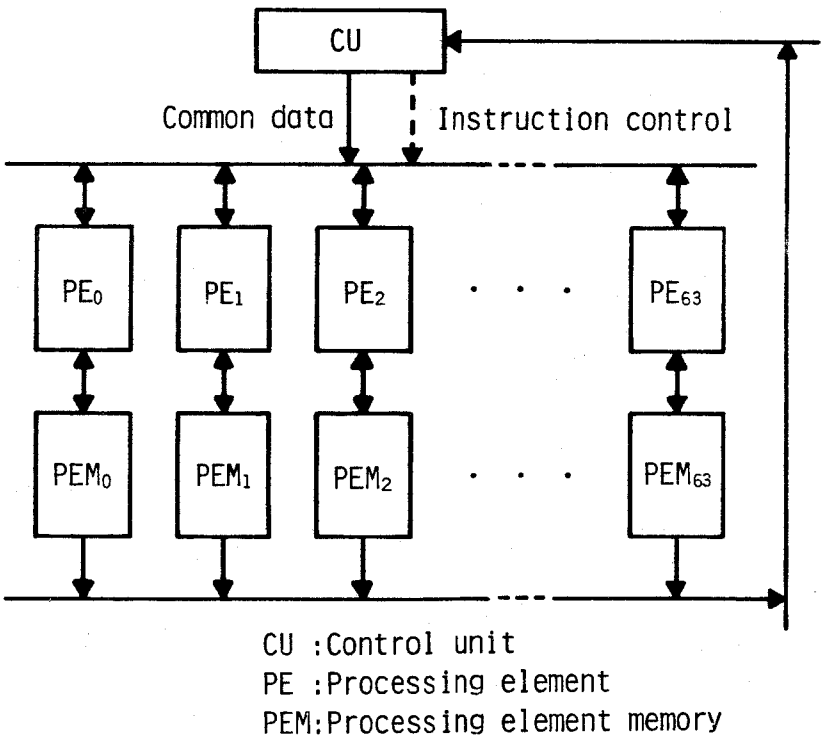


図2.1　ILLIAC Ⅳのシステム構成

Fig.2.1－System organization of ILLIAC Ⅳ.

べてのＰＥが同時に同じ加算命令を実行することによって一度に得られる。このような場合には，ILLIACⅣは，非常に効率よく動作するが，ＰＥ間通信の頻度が多くなるとその効率が低下してしまう。これは，ＰＥＭが各ＰＥに分散されているためであり，すべてのＰＥＭを一つにまとめ，共有メモリとし，すべてのＰＥのそれへのアクセスを許せば，この欠点は克服できる。しかし，この時，逆に 64 個のＰＥからのメモリアクセスの要求の調停という新しい問題が生じる。

又，ハードウェア資源の並列動作による処理能力向上手法として，図 2.2 に示すような原理に基づくパイプライン処理方式[35]がしばしば用いられてきた。例えば１次元アレイの各要素 Xi ごとに次のような多項式 Yi を計算することを考える。

$$Yi = a_6 Xi^6 + a_5 Xi^5 + a_4 Xi^4 + a_3 Xi^3 + a_2 Xi^2 + a_1 Xi + a_0$$

但し，a_0 ， a_1 ， a_2 ， a_3 ， a_4 ， a_5 ， a_6 は定数。

Yi は次のように変形できる。

$$Yi = (((((a_6 Xi + a_5) Xi + a_4) Xi + a_3) Xi + a_2) Xi + a_1) Xi + a_0$$

図 2.2 において，X_1 ， X_2 ， X_3 ， …は左方向に移動されながらパイプライン演算部に送り込まれる。まず最初のタイミングで M_1 と A_1 により，$(a_6 X_1 + a_5)$ が計算されて中間レジスタ $\triangle_1$ に入る。同時に X_1 は $\square_1$ に入り，レジスタＸでは全体が一つだけ左に移動されて， X_1 の位置に X_2 が移る。次のタイミングで M_2 と A_2 により $((a_6 X_1 + a_5) X_1 + a_4)$ が計算されて $\triangle_2$ に入る。同時に M_1 と A_1 により，$(a_6 X_2 + a_5)$ が計算されて $\triangle_1$ に入る。もちろん， X_1 は $\square_1$ から $\square_2$ に移り， $\square_1$ には X_2 が入る。レジスタＸ中では全体が一つだけ左に移動されて最初 X_1 があった位置に X_3 が移ってくる。以下同様の動作が繰返される。最初の間は A_1 ～ A_6， M_1 ～ M_6 のハードウェア資源すべてが並列に動作はしないが， Y_1 が計算されてレジスタＹに入るころからパイプライン演算部はフル稼働する。 Y_1 がパイプライン演算部から出力され始めると Y_2， Y_3 ，…と続けて結果が得られる。Ｘの要素数が多いほどパイプライン演算部の効率が良い。このようなパイプライン処理の具

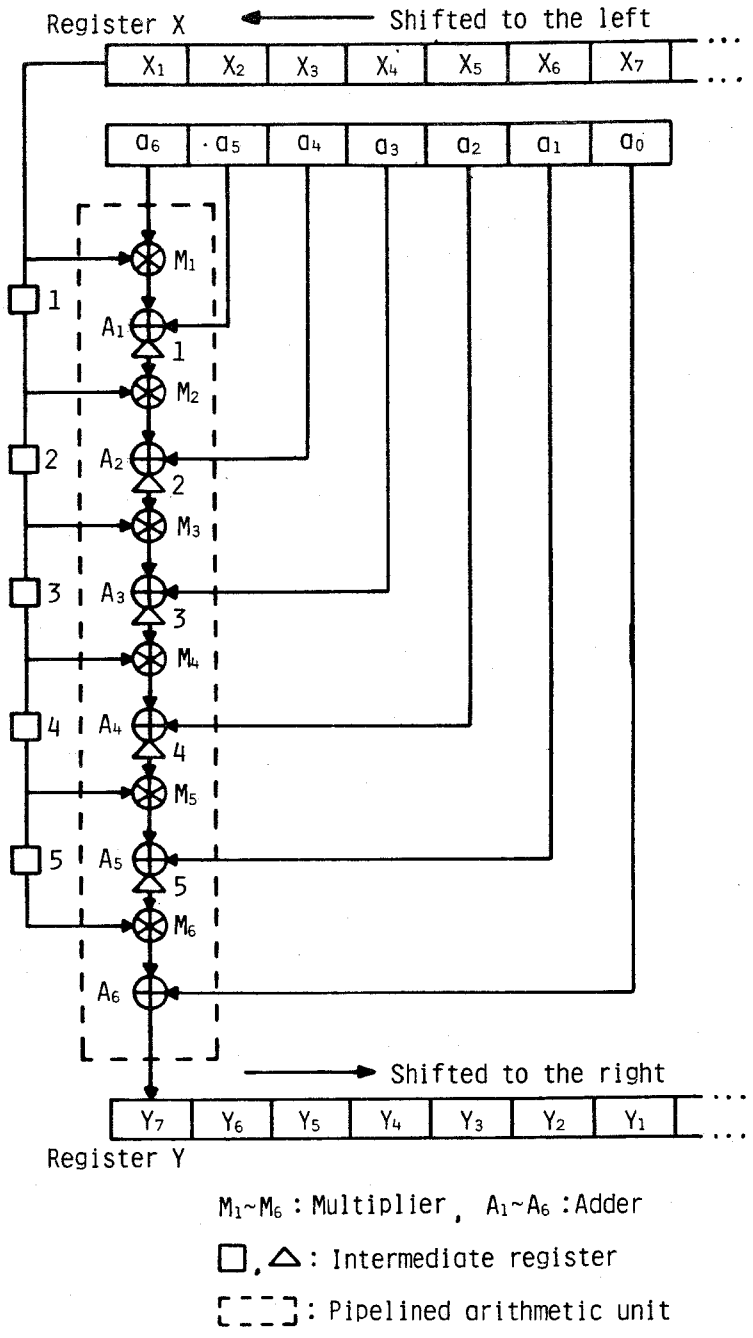


図 2.2　パイプライン形演算のためのハードウェア構成例

Fig. 2.2－An example hardware organization of pipelined operation

体的実現例は，汎用大型計算機の機械語命令の実行制御などに数多く見られる。例えば，CDC STAR100,TI ASC，CRAY-1などが挙げられる。

ここで前述したアレイ処理方式とパイプライン処理方式を比較する。N個の演算をアレイ処理方式で実行する場合，Ta・⌈N／W⌉の時間がかかる。但し，Taは命令実行時間，Wは演算ユニットの数である。これに対して，パイプライン処理方式では，Tp＋τ（N－1）の時間が必要であるが，パイプライ演算部はフル稼働すれば，実効的にはτごとに結果が得られる。但し，Tpは命令実行時間，τは各中間レジスタのラッチに要する時間である。アレイ処理方式はWの値をPEの追加によって，原理的にはいくらでも大きくできるが，PE間通信のオーバヘッドが問題となる。これに対して，パイプライン処理方式では，τの値をハードウェアの動作速度の限界まで小さくできるが，しかし，当然のことながら，命令実行過程に分岐が生じるとその性能は劣化する。この対処についてもバックトラッキング手法などの実験的検討がいくつか報告されている。

即ち，並列処理によるシステムの処理能力の向上には，究極的にはこれらの二つの技法（アレイ処理方式，パイプライン処理方式）を活用する以外には，方法がない。しかし，どちらの方式も対象とするデータ構造やシステムハードウェア構成などを基礎としているため，特定の問題については高能力を発揮するが，汎用性に欠けるため，後に定義するような高度並列処理方式の実現手法としてそのまま発展しうるものではないと考えられる。

そこで，近年の半導体技術の進歩を背景に，多数の廉価な小型のプロセッサを結合した図2.3に示すような構成をとるマルチプロセッサシステムを用いて，より自由な並列処理方式の実現手法も盛んに研究されてきた[36)-(46)]。即ち図に示すように，マルチプロセッサシステムは抽象的には，処理（Processing），記憶（Memory），通信（Communication）の各構成要素（Element）からなり，これらを横方向（Functional unit packaging）に結合すれば機能分散的，又，縦方向（Tightly coupled packaging）に結合すれば負荷分散的な並列処理が実現できる[47)]。このようなマルチプロセッサシステムの代表的な実現例として

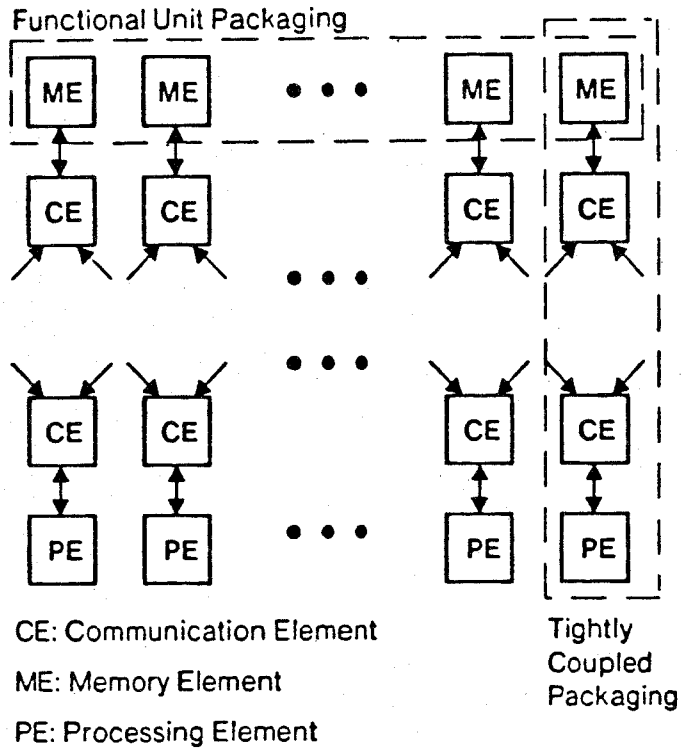


図 2.3　マルチプロセッサシステムの抽象的構造

Fig. 2.3－Abstract structure in multi－processor systems.

は，C. mmp [44]，Cm* [45],[46]などが挙げられる。C. mmp は 16 台のミニコンピュータのクロスバスイッチを介した共有メモリへのアクセスを許す密結合形のマルチプロセッサシステムである。又，Hydra と呼ばれるオペレーティングシステムや最適化機能を有するコンパイラなども同時に開発された。C. mmp は初期のマルチプロセッサシステムとしてさまざまな成果を上げたが，共有メモリ上の競合やソフトウェア構造の複雑化などについては，その問題点を提示するにとどまった。又，この経験を生かして開発された Cm* は，五つのクラスタ構成をとり，各クラスタには 14 台までのプロセッサの付与が許される。Cm* は，二つのオペレーティングシステム（Star OS, Medusa）を持ち，前者は密結合形マルチプロセッサ的動作を，後者はネットワークプロセッサ的動

作を可能とする。従って，Cm*は，マルチプロセッサシステムにおけるオペレーティングシステムや並列処理効果などのさまざまな実験に使用されている。しかし，その主要な実験結果では，並行プロセスの同期あるいはシステム資源の競合などによって，プロセッサ数に対する処理能力の向上が期待されたほど良くならないことが報告されている。従って，従来のマルチプロセッサシステムにおいても，その成功例は，特定の問題向きに専用化されたものに限定され，且つ本質的には逐次処理の影響を脱しない方式[47]であり，次のように定義される高度並列処理方式の実現手法として，そのまま発展しうるものではないことがわかる。

〔定義2.2〕 並列処理方式のうち，対象とする問題に内在する並列性が許す限りシステム資源の投入に見合う処理能力向上を可能とするものを高度並列処理方式とする。

〔定義2.1〕と〔定義2.2〕との比較からわかるように，前者は対象とする問題あるいはジョブの並行して動作可能なプロセスへの分割からその処理方式が決定される。ここで，その分割過程で使用可能なハードウェア資源の有効利用という判断が介在することは言うまでもない。これは，ハードウェア資源は高価で且つ貴重であると言う前提から出発している。事実，従来の計算機システムは常に高価なハードウェアがまず存在し，そのプログラミングが後を追う形で発展してきた。これがいわゆるソフトウェア危機を招いた重要な原因の一つでもあった。しかるに一方，〔定義2.2〕は対象とする問題に内在する並列性を優先し，その有効利用をめざしている。即ち，このような高度並列処理を実現する方式は，1）対象とする問題に内在する並列性を余すことなく最大限に活用できる自由な機能および負荷分散処理方式，更に，これらの多数の処理を制御あるいは駆動するため，2）分割損のない効果的な非同期分散形の制御原理，並びに3）システムと外界との効率の良い入出力インタフェース，を持たねばならない。まず自由な機能および負荷分散処理方式の実現には，同時並行的に実行される副プロセス及びそれら副プロセス間の並列処理構造が明確に定義されなければならない。次に，分割損のない分散制御の実現には，システ

ムの稼働率あるいは信頼性を保証できる代替可能でモジュラな制御機能による，システムの物理的構成規模やその資源割当から独立な分散形制御原理が必要である。更に，効果的な外界との入出力インタフェースの実現には，イベント駆動形計算原理およびそれによるストリーム形処理を可能とする動作モードがシステムに必要となる。ここで，これらの要件を満たすシステムの一貫性を保証するには，システム全体の機能あるいはプログラムを副作用なく記述できるツールが本質的に必要となる。既に述べたように，並列処理方式実現上の最大の問題点はそのプログラミングが必ずしも簡単でないことであった。従来のハードウェア先行形の実現手法をとれば，現状のソフトウェア危機を一層助長することは明白である。そこで，本研究では，高度並列処理方式の実現手法として，アルゴリズムに内在する並列性を余すことなく記述可能な並列処理言語を基礎として，その言語の実行制御原理の実現によりシステムの構成をめざす，即ち，言語主導形の実現手法を採用した[27)-(31)]。

2．3　D^3Lの必要性とその特徴

高度並列処理の実現には，同時並行的に実行される多数の処理の間の相互干渉，いわゆる副作用のないことが前提条件になる。従って，この条件を満たすプログラムの記述法を確立しなければならない。又，並列処理構造の記述では，了解性の向上も重要な課題である。単一の在来形プロセッサ上での実時間制御プログラムの設計上の困難と複雑さを見れば，多数のプロセスを調停しなければならない並列処理システムのプログラムの組織化などは至難であろうといった見方が一般的であった。プログラムの考え方を本質的に変えない限り，このような見方はむしろ極めて当然の結論と言わざるを得ない。更に，了解性，即ちプログラミングの容易さは，単に言語の記述能力だけではなく，システム構想を仕様化する段階から始まり，設計・評価・改良・保守など，そのシステムの着想から実現に至る全期間を通して評価されねばならない。ことに最近は，設計段階の効率化と期間短縮は言うまでもなく，システム完成後の小仕様変更を含む，保守あるいは改良の容易性が重要視されている[48),(49)]。

これらの要求を満たすには，なるべく補足的説明を加えないで，高度並列処理プログラムを了解しうる記述手法が必要である。つまり，高度並列処理方式実現の核となる言語としては，従来のドキュメントとプログラムとが一体化した表現形式あるいはドキュメントそのものが既にプログラムとして機能する表現形式の確立を目標としなければならない。従って，本研究ではプログラムの表現形式の選択が極めて重要となる。

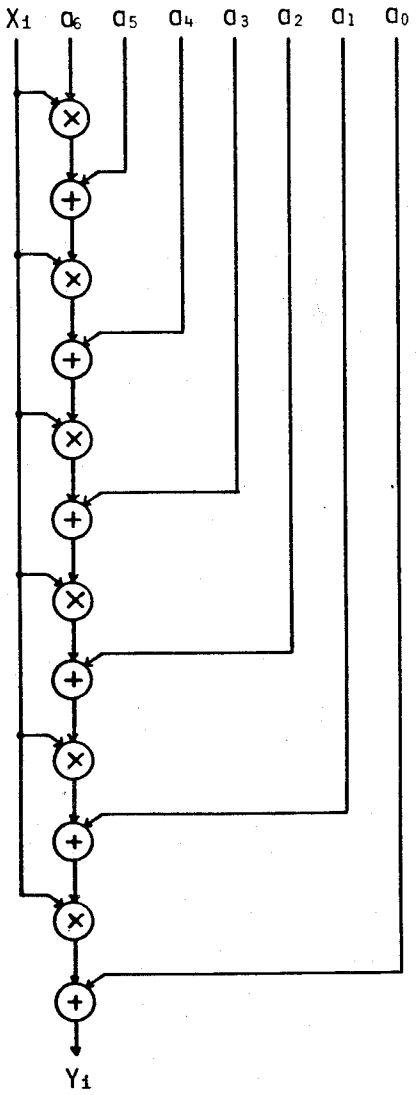
並列処理の記述法に関して最近多くの提案がある。例えば，単一割当言語[9],[10]は，プログラム中の各変数の定義を高々一回に制限し，プログラム中に陰に含まれる並列性の検出と検証性の向上を意図している。しかし，この方法は，本質的に逐次処理の概念を伴う文章形の代入文を用いるので，並列処理の了解性に欠ける。又，データフロー言語[11],[12],[15],[16],[20],[22]-[24]は，次に定義するような各処理のデータ従属性を示す図式に基づいた記述形式を採用すると共に，その図式上のトークンの流れに対してデータ駆動原理を適用して，非同期・分散形の並列実行をイメージすることによる，並列性の自然な表現をめざしている。

〔定義2.3〕 ノードとアークで構成される有向グラフのうち，ノードに処理を割当て，その処理に付随する入出力をノードに入るアーク及びノードから出るアークに対応させ，各処理の入出力関係に基づいたアークの結合により生成されるものを，データ従属図式（Data-dependence schema ; DDS）とする[11]。

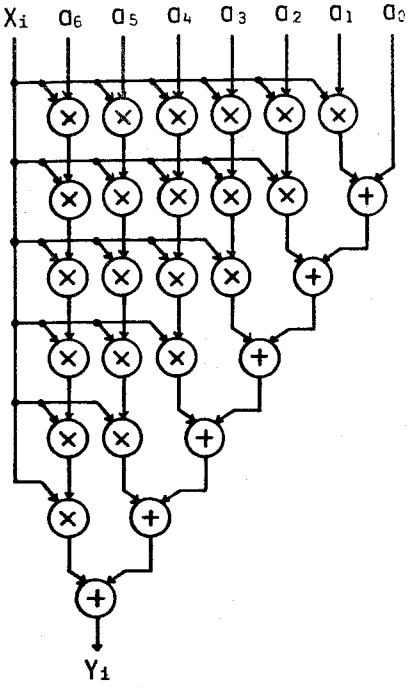
〔定義2.4〕 ＤＤＳにおいて，各アーク上に割当てられたデータの流れをイメージする時，そのデータの存在を示すものをトークンとする[11]。

〔定義2.5〕 ＤＤＳにおいて，各処理の実行をそのノードに入るすべてのアーク上にトークンが揃った時に可能とする制御原理をデータ駆動とする[11]。

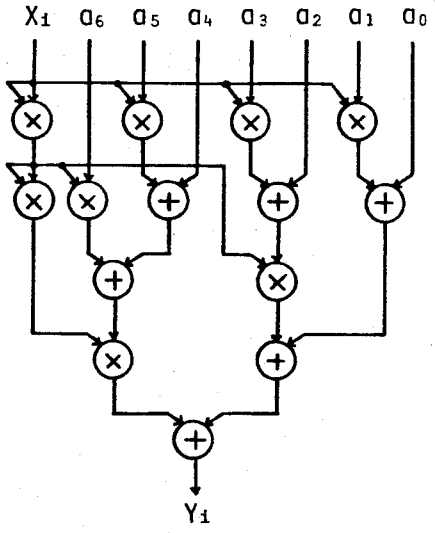
しかし，このＤＤＳに基づくデータフロー言語は記憶の概念を持たないため，履歴依存性[26]を有する処理を直接的には表現できない。しかしながら，ＤＤＳは，文章形記述には求められない本質的な利点を持っている。図2.4(a)に2.2に述べたパイプライン処理方式のＤＤＳを用いた記述を示す。又，図2.4(b)及び(c)にこの多項式の最も自然な解法，より速い解法をＤＤＳを用いて示す。即



(a) $Yi=(((((a_6 Xi+a_5)Xi+a_4)Xi+a_3)Xi+a_2)Xi+a_1)Xi+a_0$



(b) $Yi=a_6 X_i^6+(a_5 X_i^5+(a_4 X_i^4+(a_3 X_i^3+(a_2 X_i^2+(a_1 X_i+a_0)))))$



(c) $Yi=(a_6 X_i^2+(a_5 X_i+a_4))X_i^2 X_i^2+((a_3 X_i+a_2)X_i^2+(a_1 X_i+a_0))$

図 2.4　データ従属図式によるアルゴリズムの記述

Fig. 2.4－Algorithm descriptions in data-dependence schema.

ち，ＤＤＳは記述形式そのものにアルゴリズムに内在する並列処理構造（同時並行，パイプライニング）を陽に示している。更に，図式表現の持つ二次元的表現能力は並列処理の記述法として，逐次記述手法には求められない，本質的な利点を持っている。日常生活にしばしば利用される組織図などの図式表現を文章による逐次的記述と比較すれば，その了解性の差は歴然としている。又，システム記述の領域でも，ブロック図あるいは論理回路図など並列動作構造の了解性に優れた表現手段が広く用いられている。しかしながら，本研究の立場は，文章記述を全面的に否定するものでは決してない。少なくとも高度並列処理の自然な記述を目標とする以上，同時並行動作を陽に記述しうる図式表現を許しこれを実用的な方式に発展させる方法が最適であると考える。

本研究のように，ハードウェアと原則的に独立に言語を検討する方針をとれば，言語の選定および設計の自由度が極めて大きくなる。つまり，既に存在する機械を意識して表現法を考えるのではなく，先に述べたプログラミングの容易さを主眼として，最適の表現形式ができるからである。ハードウェアは最後に，この言語を最も効率よく実行するように，設計すれば良い。この立場からも，言語とその記述法の選択は本研究の基礎として極めて重要な意味を持っている。しかし，図的表現が計算機言語として機能するためには，図式の厳密な生成，解釈および実行の規則が必要となる。本研究に提案する図的データ駆動言語（Diagrammatical Data-Driven Language ; D^3L）[27),(28),(30)]は，後述の利用者言語と物理的並列処理機構との間に位置する中間言語として位置づけられる。これは，現状のソフトウェア危機を背景にした仕様記述の高級化，並びに上述したような高度並列処理方式における副作用の回避のための厳密なプログラム記述という背反する要件を考慮したためである。即ち，D^3Lは本研究の理論的基礎としての高度並列処理方式における検証性（プログラムの正しさの保証）を与えることを主眼として設計されている。以下に，D^3L の特徴を簡単に述べる。

(a). D^3L の基本構造　D^3L の基本図的表現では処理をノードで表現し，その処理に必要なデータの組をそのノードに入るアークの組で表している。各

処理は入力アークのすべてにデータが存在する時かつその時に限り実行（発火）可能となる。この限りでは，データ駆動言語はデータフロー言語と全く同じ，即ちデータ依存性のみに支配される動作をする。D^3L では更に，単なる各処理の連接のみならず，プログラミング言語として基本的に必要となる選択および再帰を示す制御構造，データフロー言語では直接的に表現不可能な履歴依存処理の記述においても，トークンの流れに関して厳密にこのデータ駆動原理に従って実行される。この原則のために，この図式で表現されたプログラム内のすべての処理は，他の処理の状況と無関係に各ノードの発火条件だけに着目して，それぞれの処理を独立に開始してよいことが保証される。つまり，同時に発火条件を満足した複数のノードが並列に処理を実行しても相互干渉を起こさない保証が図式に組み込まれているので，高度並列処理の記述が可能となる。

(b). D^3L のブロック構造　D^3L では，いくつかのノードを含む図式は一つのブロック（手続）を形成する。これら各ブロックの入力，出力および後述の履歴依存の表明が，前述のノードに対するのと同様に，図的に明快に記述される。即ち，D^3L プログラムは，厳密に定義された構成規則に従って生成され，一定の解釈に従って実行されるので，その正当性が保証されている。従って，D^3L では記述されるプログラム全体が厳密な実行規則を持ち，階層的なブロック構造として表現される点に大きな特徴がある。

(c). D^3L の検証性　一般にプログラムの正しさは，それ自体独立に議論できるものではなく，あくまである定められた仕様に対して論議されねばならない。例えば，従来の検証方法では，プログラムの仕様を入力表明と出力表明の組として表現し，入力表明を満たす正当な任意の入力に対してプログラムを実行した時，プログラムが確かに停止しかつ出力表明を満たす正当な出力が得られる場合にそのプログラムを全正当であると定義している。しかし，D^3L では，従来の逐次実行形式のプログラムと異なって，全ての処理が処理資源の許す限り非同期に並列実行されるので，プログラムの正当性の検証は一見より困難になるように考えられる。しかしこの図式では，プログラムの基本的な制御構造を，連接，選択および再帰に限定し，図式中のノード間の半順序関係を保

証しているので，次の要件を満たすことで容易にその正しさを確認できる。

① D^3L プログラムに，正当な入力組を与えた時，有限の実行過程によって，正当な出力組に到達し停止すること。

②実行終了後において，D^3L プログラムにデータの滞留がないこと。

③実行過程において，すべての処理が実行されること。

(d). D^3L の履歴依存処理　D^3L では更に次の構成規則および解釈を付加して，図的表現の明快さと言語の検証性を保証しながら，履歴依存性を有する処理の記述を可能にしている。

①履歴依存性を有するノード及びそれらへのアーク

②履歴依存性を有するノードの実行順序を決定できる制御構造

③履歴依存性を有するノードの実行規則

D^3L は，前述の通り，データフロー図式の履歴依存処理に関する欠陥を補い且つ厳密な実行規則に従って高度並列処理を表現する，中間言語である。即ち，D^3L は物理的処理機構から独立な最低レベルの言語として位置づけられ，この言語から下位の各種の物理的な高度並列処理機構上の実行形式への変換は容易である。又，D^3L から上位に，D^3L の文法規則を土台として階層的に各種の利用者言語体系を組み立てることも可能である。更に，この言語の階層的記述能力を利用すれば，同一の処理機構の繰返し構造の物理的システムが同じ言語で統一的に記述できるので，更に大規模な並列処理システムが容易に構築できることも大きな特徴となっている。

更に，D^3L を基礎に組み立てられた言語体系は，D^3L の特徴をそのまま保存するので，次のような特徴を付与される。

①．検証性：プログラムが言語の解釈に従って実行された場合の結果に非決定性がないこと。

②．階層性：プログラム記述階層（レベル）の自由な設定により複雑なプログラムの構造化ができること。

③．了解性：プログラムの並列処理構造が陽に示されること。

これらの性質はすべて従来のプログラム技法上大きな障害となっていた基本

的問題の解決に役立つものである。又，D^3L を基礎とする言語，即ち具体的な並列処理機構を示す実行形式あるいは問題向きの利用者言語は，記述対象の持つ並列処理の論理的構造をそのまま陽に反映した図的表現を生成するので，非常に見通しの良いプログラムの作成あるいはシステムの構成手段となることが期待される。

(a). D^3L の実行形式　中間言語としての D^3L は，各種の物理的並列処理機構に翻訳されて実行され，複数のアーキテクチャを生み出す。これは，在来形の機械が，同じ範疇に属する言語を使用しながら，いくらか異なったアーキテクチャを持つのと同様である。高度並列処理アーキテクチャでは，D^3L で図式表現された処理を最も高速に実行しうる性質が重要な条件となる。従って，D^3L は実行形式の検討の基礎としても重要な役割を果たす。

(b). 問題向き利用者言語　D^3L 記述は，従来のプログラミング言語に比較して，かなり見易いものである。特に高度並列処理の表現については，極めて了解性に優れたプログラムを生成することが，具体例の記述でも確かめられている[50]。しかし，中間言語はいわゆる機械語に近い低レベルの言語であるため，そのまま一般の利用者に提供するにはなお不充分である。本研究では更に高級な表現能力を持ち，利用者が日常使用する記述法になるべく近い表現法を利用者に提供すべきであると考えている。この表現法のサポート機能は，一種の設計援助システムにあたると考えられる。図的中間言語を採用すれば，利用者言語もまた図的表現をとるのが最も自然である。言うまでもなく図的表現はしばしば設計段階で利用される手法であり，構造記述手法としてむしろ望ましい方法である。例えば論理設計における，概念的な論理ブロック図に始まり詳細な論理回路図に至る，トップダウン的仕様記述手法はその好例と考えている。

2.4　結　　言

本章では，高度並列処理方式の実現手法として，従来のハードウェア先行形に代わる，図的中間言語を核とする言語主導形の開発手法について述べた。まず従来の並列処理システムにおける本質的な問題点がそのプログラミングの困

難さにあることを示すと共に，高度並列処理方式に求められる要件を明らかにした。次に，高度並列処理の実現手法における中間言語として採用した D^3L の特徴を簡単に述べ，これが物理的並列処理機構の模索のみならず，問題向き利用者言語あるいは仕様記述過程の定式化に対しても有望であることを示した。

一般的な利用者言語の形態とその設計手法には未解決の部分が多いが，既に論理システム設計あるいはこれに帰着させうる応用分野については見通しが得られている。図的データ駆動言語の階層的性質を利用すれば，機器設計のみならず事務処理の記述におよぶ，種々の仕様記述に適した多様な利用者言語を開発しうる可能性が充分認められる。現在，以下の要件を満たす言語仕様を持つ，階層的利用者言語の導入を検討中である。

①データ駆動図式に比べてより一般的に利用されている，ブロック図などの階層的な図式記述法を採用すること。

②コンパイルなど機械的手段によって D^3L に変換可能な構成規則および解釈を具備していること。

③対象とする応用分野におけるプログラム設計上の規則，制約条件などが自動的に反映されるインタラクティブなインタフェースを持つこと。

これらの要件，特に②及び③については残されている課題が多い。現在のところ，次のような検討を行っている。

①図式中に陰に含まれる履歴依存性の検出とそれを中間言語としての D^3L の仕様に変換する手法

② D^3L への抽象データ構造とその記述法の導入

以上2点が解決すれば，ある程度の曖昧さを許した仕様記述が可能となるので， D^3L の記述能力は飛躍的に向上する。従って，利用者言語を D^3L に変換するコンパイラの作成が可能となるのみならず， D^3L を高度並列処理方式における強力なシステム記述言語として発展させられる可能性もある。

第３章　履歴依存性を許すデータ駆動図式

3.1　緒　　言

本章では，履歴依存性を含めた並列処理構造を記述形式に陽に反映し，且つ副作用を生じない階層的な並列処理記述法を実現するために，D^3L に採用した履歴依存性を許すデータ駆動図式について述べる[27],[28],[30]。本図式は，図式中に履歴を示すノードの記述を許すが，これによって生じる非決定性を回避可能なデータ駆動図式（Data-driven schema）である。更に，この図式で記述される階層的あるいは再帰的なプログラムには，厳密なデータ駆動原理に従って実行され且つ副作用を生じない保証が与えられている。即ち，本データ駆動図式は，階層的あるいは再帰的にデータ駆動副図式（Data-driven sub-schema）を定義することによって生成される。本データ駆動副図式は，入出力表明，関数的処理，階層的あるいは再帰的手続，条件判断などの処理内容を示すノード，及びデータと制御の流れ，コピー，マージを示すアークから構成される有向グラフである。又，本データ駆動図式は，制御構造を連接，選択および再帰に標準化することにより，記述されるプログラムの構造化をはかると共に，ノード間のデータ従属性に関する半順序関係を保証している。従って，各ノードに割当てられた処理の実行順序は，アークによって示されるデータ従属性だけで決定されるので，トークンの流れに関する限り，厳密にデータ駆動の原則に従う。更に，記述されたプログラムには，副作用のない実行を保証する検証性が付与されている。

以下，本章では，まず本データ駆動図式の基本的な仕様を述べ，記述対象とするプログラムを自然に構造化する制御構造の採用によって，副作用の回避が保証されることを示す。次に，本データ駆動図式への履歴依存性の記述法の導入およびこれによる副作用の回避の原理を述べる。最後に，図式中の履歴の具体的実現例として，タグ付き記憶セルを示し，これを副作用なく参照・更新するための規則を提案する。

3．2　データ駆動図式

本節ではデータ駆動図式の基本的な仕様について述べる。まず本図式の生成規則とその解釈および実行規則について述べる。次に，制御構造として，連接選択，再帰を用いて記述されたプログラムが副作用なしであることを等価な実行系列を持つペトリネットにおける到達可能木を用いて示す。

3．2．1　データ駆動図式の仕様

【定義3．1】 データ駆動図式 S_0 は，図3.1に示す要素を用いて，次の手続によって記述される。

1° 与えられた問題を1つのブロック（Block）として，図3.1(a)に割当て，その内容を図3.1(b)～(j)で構成されるデータ駆動副図式 S_0' で記述する。

2° もし，S_0' がいくつかの未定義な図3.1(j)を含む場合，それぞれを1つのブロックとして，図3.1(a)に割当て，その内容を S_0' で記述する。

3° もし，すべての S_0' が未定義な図3.1(j)を含まない場合，与えられた問題の記述は終了する。それ以外の場合は2°へ戻る。

従って，S_0 は階層的あるいは再帰的なブロック構造を持つ。

【定義3．2】 S_0 の解釈 I_0 は次のようである。

①解釈の領域として空でない集合D，$D = V \cup \{ T, Nil, null \}$ をとる。
但し，Vは値の集合，T及びNilは真理値，nullは空値を示す。

②各アークには，次のdあるいはcが割当てられる。

(i)　データ（Data）：$d \in V \cup \{ Nil, null \}$

(ii)　制御（Control）：$c \in \{ T, Nil, null \}$

null以外のd，又はcを割当てられたアークはトークンtを持つと解釈し，$W = V \cup \{ T, Nil \}$ をトークンの集合とする。

③コピー（Copy）及びマージ（Merge）はdが割当てられ，次のようにtを推移させる。

(i)　コピー　$C : W \rightarrow W^S$

$C(t) = (t_1, \cdots, t_S), \ t_1 = \cdots = t_S = t$

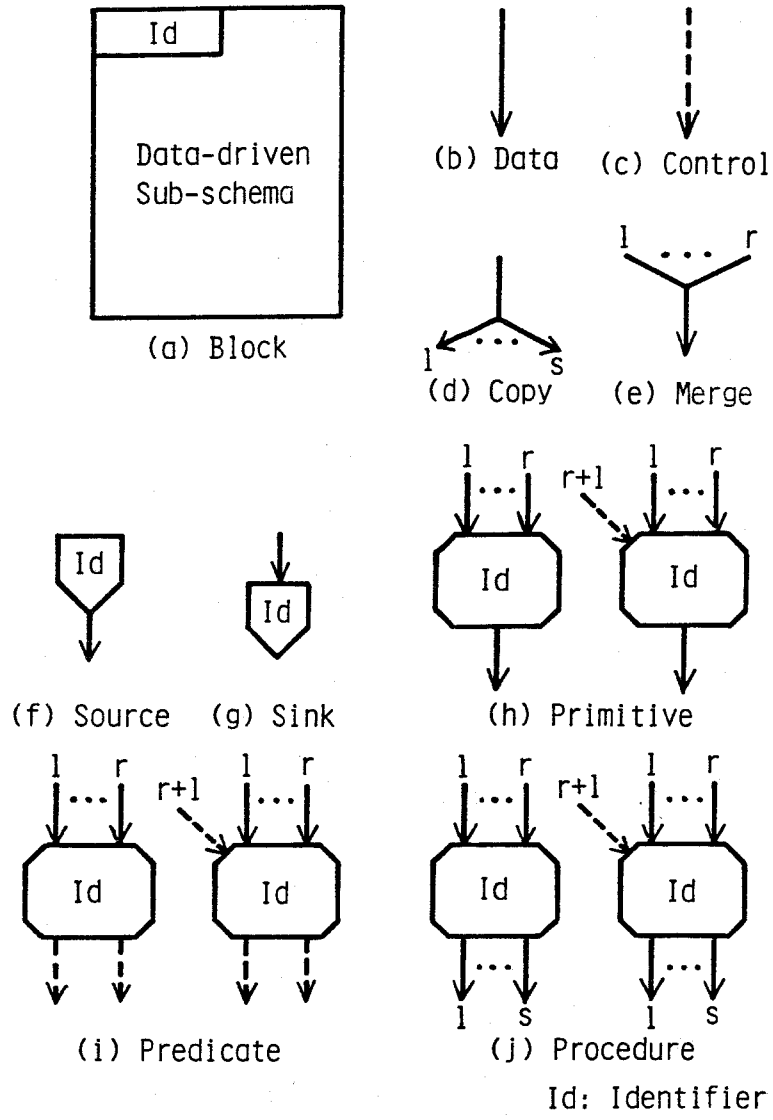


図 3.1　データ駆動図式の要素

Fig. 3.1-Elements of the data-driven schema.

(ii) マージ　$M : W^r \rightarrow W$

$M(t_1, \cdots, t_r)$

$$= \begin{cases} t_i & (\text{唯一つの } t_i \text{ が } v, \text{ 他の } t \text{ はすべて Nil}) \\ Nil & (\text{上記以外}) \end{cases}$$

ここで $(t_1, \cdots, t_s) \in W^s$, $v \in V$

④ソース（Source）及びシンク（Sink）は，次のように外界および他の S_0' と t を送受する。

(i) ソースは，S_0' の始点を示し，受取った t を，そのデータアーク上に生成する。ここで，生成とは null を t に置換することである。

(ii) シンクは，S_0' の終点を示し，そのデータアーク上の t を消費し，それを送出する。ここで，消費とは t を null に置換することである。

⑤プリミティブ（Primitive）には，次のものを割当てる。

(i) 関数　$F : W^r \rightarrow W$

$F(t_1, \cdots, t_r)$

$$= \begin{cases} f(t_1, \cdots, t_r) & (t_i \in V,\ i = 1, 2, \cdots, r) \\ Nil & (\text{上記以外}) \end{cases}$$

但し，$f : V^r \rightarrow V$

(ii) 条件付関数　$F' : W^{r+1} \rightarrow W$

$F'(t_1, \cdots, t_{r+1})$

$$= \begin{cases} F(t_1, \cdots, t_r) & (t_{r+1} = T) \\ Nil & (t_{r+1} = Nil) \end{cases}$$

⑥プレディケート（Predicate）には，次のものを割当てる。

(i) 述語とその否定の組　$P : W^r \rightarrow \{T, Nil\}^2$

$P(t_1, \cdots, t_r)$

$$= \begin{cases} (T, Nil) & (p(t_1, \cdots, t_r) \text{ が真}) \\ (Nil, T) & (p(t_1, \cdots, t_r) \text{ が偽}) \\ (Nil, Nil) & (\text{上記以外}) \end{cases}$$

但し，$p : V^r \rightarrow \{T, Nil\}$

(ii) 条件付述語とその否定の組　$P' : W^{r+1} \longrightarrow \{T, Nil\}^2$

$$P'(t_1, \cdots, t_{r+1}) = \begin{cases} P(t_1, \cdots, t_r) & (t_{r+1} = T) \\ (Nil, Nil) & (t_{r+1} = Nil) \end{cases}$$

⑦プロシージャ（Procedure）には，次のものを割当てる。

(i) 手続　$Proc : W^r \longrightarrow W^s$

(ii) 条件付手続　$Proc' : W^{r+1} \longrightarrow W^s$

但し，$t_{r+1} = T$の時 $Proc' = Proc$

$t_{r+1} = Nil$の時 $Proc'$の出力はすべてNil

Procの内容はその識別子（Identifier）の一致するブロックに含まれるS_0'で示され，その解釈は①～⑦に従う。

【定義3.3】 データ駆動図式S_0とその解釈I_0との組$\langle S_0, I_0 \rangle$をデータ駆動プログラムDP_0とする。

【定義3.4】 DP_0は次のように実行される。

① DP_0の入力となるすべてのソース，即ち，第一階層のブロックに割当てられたS_0'に含まれるすべてのソースにtが与えられると，DP_0の実行が開始される。

②必要なtが揃ったノードが，すべてのtを消費し，その解釈に従ったtを生成することによってDP_0の実行が進行する。但し，プロシージャは，その識別子の一致するブロックに割当てられたS_0'に含まれるソースにtを与え，$\langle S_0', I_0 \rangle$を実行し，シンクに受取られたtを生成する。

③ DP_0の出力となるすべてのシンク，即ち，第一階層のブロックに割当てられたS_0'に含まれるすべてのシンクにtが受取られると，DP_0の実行が終了する。

3.2.2 制御構造と検証性

DP_0は非同期に並列実行されるので，従来の逐次形プログラムよりも実行順序における自由度が大きく，実行結果に非決定性がないことを確認できるよ

うな検証性について考慮されなければならない。

逐次形プログラムでは，その構造化のために，制御構造を標準化し階層的に記述することが一般に行われている[51],[52]。ＤＰ₀においても，その図的表現の明快さに由来する了解性を保証できるような階層的記述および標準化された制御構造の採用が当然必要である。

逐次形言語の図的記述であるフローチャート図式については，次のことが既に知られている。

〔性質3．1〕 フローチャート図式によるプログラムの制御構造は，連接，選択，繰返しだけで記述できる[53]。

又，データフローモデルの図的記述であるデータフロー図式（ＤＦＳ）によるプログラム（ＤＦＰ）の実行結果に非決定性が生じない保証，いわゆる副作用の回避については，以下のことがわかっている。

〔定義3．5〕 図3.2のように，内部にトークンを含まない状態M_0にあるＤＦＰに，入力トークンX_1，…，Xmを与えた時，すべての入力トークンを消費し，有限の実行系列により，その入力トークンに対して一意的な出力トークンY_1，…，Ynを生成した後，再びM_0に戻る時，このＤＦＰは，入力トークンX_1，…，Xmに関して副作用なし（No side-effect）である[11]。

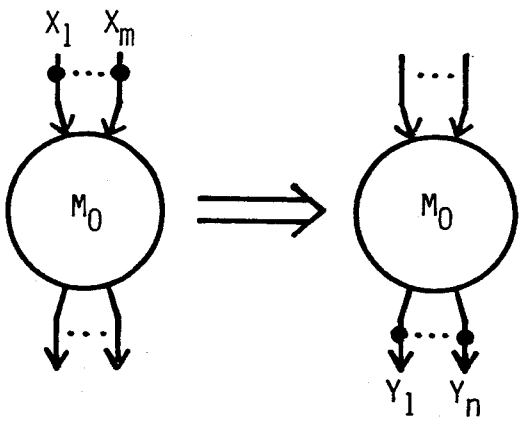


図3.2　副作用のないデータフロープログラム

Fig. 3.2－Data－flow program without any side-effects.

〔定義3．6〕 関数的なオペレータを非巡回的に結合して記述されたＤＦＰは良形（Well-formed）である[11]。

〔性質3．2〕 良形なＤＦＰは，それを構成する関数群の定義域内の入力に関して副作用なしである[r1]。

定義3.5より，副作用なしのＤＦＰはそれ自身入力トークンX_1，…，X_mから出力トークンY_1，…，Y_nを生成する一つの関数的なオペレータとみなすことができる。従って，明らかに次の系が成立する。

〔系3．1〕 良形なＤＦＰを表現するＤＦＳ中のノードを，副作用のないＤＦＰを表現するＤＦＳで置換することによって生成されるＤＦＰは良形である。

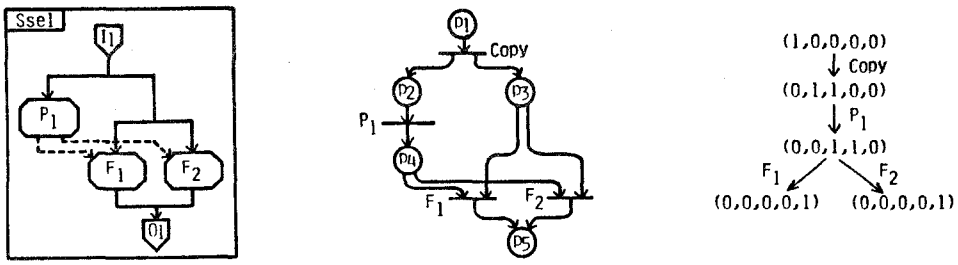
S_0によるプログラムは，定義3.1に従ってS_0'で記述される階層的あるいは再帰的なブロック構造を持つ。ここで，図的表現の二次元的表現能力を活用して明確にS_0'の制御構造を標準化する。但し，これ以降の図では，簡単のためノードの入力数を最小にしている。

(1)連接：ノードとして，ソース，シンク，プリミティブ及びプロシージャだけを用い，各ノードをコピー及びデータアークで非巡回的に結合したデータ駆動副図式 Scon。

(2)選択：図3.3(a)に示すように，プレディケート（P_1）の実行結果に従って，プリミティブ（F_1，F_2）へ排他的にT及びNilが生成されるデータ駆動副図式 Ssel。但し，F_1，F_2をプロシージャに置換することにより，生成されるデータ駆動副図式 Ssel′を許す。

(3)再帰：図3.4(a)に示すように，プレディケート（P_1）の実行結果に従って，Srec の再帰呼出しの続行あるいは終了が選択されるデータ駆動副図式 Srec。但し，Srec をSrec を含むプロシージャに，F_1，F_2，F_3を Srec を含まないプロシージャに置換することにより生成されるデータ駆動副図式 Srec′を許す。

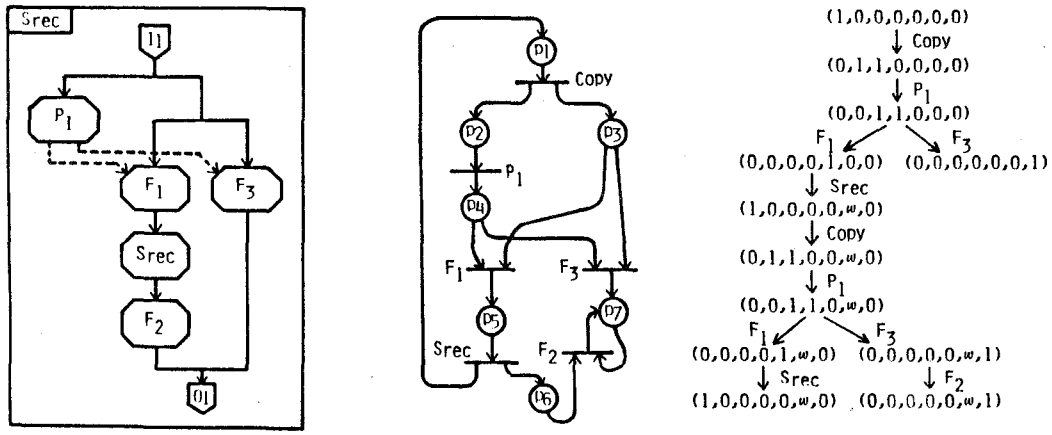
従って，選択・再帰を示すS_0'においても，トークンの流れに関しては，連接を示すS_0'と同様である。又，繰返しを禁止し再帰を採用することにより，S_0におけるノード間の半順序関係が保証される。



(a) Conditional sub-schema: Ssel.　(b) Petri-net graph.　(c) Reachability tree.

図3.3　選択構造を持つデータ駆動プログラム

Fig. 3.3－Conditional data－driven program.



(a) Recursive sub-schema : Srec.　(b) Petri-net graph.　(c) Reachability tree.

図3.4　再帰構造を持つデータ駆動プログラム

Fig. 3.4－Recursive data－driven program.

〔定義3.7〕 S_0 の中で，そのすべての S_0' の制御構造が Scon，Ssel，Ssel′，Srec，又は Srec′のいずれかであるデータ駆動図式 S_1 とその解釈 I_0 との組〈 S_1 ， I_0 〉をデータ駆動プログラム DP_1 とする。

〔性質3.3〕 Ssel とその解釈 I_0 との組〈 Ssel， I_0 〉で定義されるデータ駆動プログラム DP_2 は，解釈の領域D内の入力に関して副作用なしである。

（**証明**）DP_2 は，図3.3(b)に示す等価な実行系列を持つペトリネットグラフ（Petri-net graph ；PNG）[54] に変換できる。即ち，ソース I_1 は p_1，シンク O_1 は p_5，データアークは p_2 ～ p_4 の各場所に，プリミティブ F_1，F_2 及びプレディケート P_1 は同図中に示す遷移にそれぞれ対応する。又，P_1 による F_1，F_2 の選択的実行は，場所 p_3，p_4 におけるトークンの衝突[54] により表される。従って，PNGのトークンの集合は｛v，T｝となる。このPNGの場所 p_1 に対する初期マーク付け（1,0,0,0,0）からの到達可能木（Reachability tree ；RT）[54] を図3.3(c)に示す。RTは，PNGのマーク付けを示す節点と遷移の発火により生じうる，可能なマーク付けの変化を示す枝を持つ木である。従って，同図より，このマーク付けPNGは，有限の実行系列によりシンクに対応する場所 p_5 に対する最終マーク付け（0,0,0,0,1）に到達する。故に，DP_2 は副作用なしである。

（証明終）

〔性質3.4〕 Srecとその解釈 I_0 との組〈Srec，I_0〉で定義されるデータ駆動プログラム DP_3 は，有限回の再帰呼出しによりその実行が停止するような解釈の領域D内の入力（以降，正当な入力と呼ぶ）に関しては，副作用なしである。

（**証明**）DP_3 は，DP_2 と同様な方法で図3.4(b)に示す等価な実行系列を持つPNGに変換できる。ここで，Srec の再帰呼出しによる繰返し実行は遷移Srec からソース I_1 に対する場所 p_1 へのループで，繰返し回数は場所 p_6 に蓄積されるトークンの数でそれぞれ表される。このPNGの初期マーク付け（1,0,0,0,0,0,0,）からのRTを図3.4(c)に示す。ここで，場所 p_6 における ω は，再帰呼出しの回数により任意に大きくなる数を示す。即ち，p_6 におけるトークンは，Srec の実行によりの生成され，F_2 の実行により消費される。従って，正当な入力に対しては，このマーク付けPNGは，有限の実行系列により，シンクに対応する場所 p_7 に対する最終マーク付け（0,0,0,0,0,0,1）に到達する。故に，DP_3 は副作用なしである。

（証明終）

【性質3．5】 DP_1 は正当な入力に関して良形である。

（証明）定義3.5より明らかに，S_1 はすべてのプロシージャをそれに割当てられたデータ駆動副図式 S_1' に置換することによって，再帰を示すプロシージャ以外のプロシージャを含まないデータ駆動図式 S_4 に展開できる。S_4 は，系3.1，性質3.3，性質3.4に基づき，次の手続によって，ソース，シンク，プリミティブおよび関数的なオペレータの連接を示すデータ駆動図式 S_5 に変換できる。

1° S_4 に含まれる，SselおよびSrecを関数的なオペレータに置換する。

2° S_4 にプレディケートが含まれていなければ終了。もし含まれていれば1°に戻る。

プリミティブは関数的なオペレータであり，また明らかに S_5 の接続構造は非巡回的であるので，定義3.6より，S_5 と解釈 I_0 の組〈S_5，I_0〉で定義される DP_5 は良形である。故に，DP_1 は良形である。

（証明終）

性質3.2及び性質3.5より明らかに次の系が成立する。

【系2】DP_1 は正当な入力に関して副作用なしである。

即ち，DP_1 では，第一階層のソース群によって示される入力表明を満たす正当な入力に対するプログラムの実行が停止する時，第一階層のシンク群によって示される出力表明が満たされることが保証される。更に，S_1 に全域関数でないfを伴うプリミティブが割当られている時も，それが生成したNilは保存され，DP_1 の出力にNilが生成されるので検出可能である。

3．3 履歴依存性の導入

従来のデータフローモデルは，その検証性への考慮のみならず並列性を最大限に引出すために，非同期に並列実行される各処理は，副作用がない関数的なオペレータであることを前提にしている。従って，複数の処理により共有され，参照および更新が可能な記憶セルという概念はなく，元のデータを処理した結果は，常に新しいデータとして，別に生成されなければならない。このため，

従来記憶セル中のデータの参照および更新によって表現していた履歴依存性を持つ処理をデータフローの概念に基づいて実行するためには，新たなデータを次々と生成し，その生成順序を何らかの形で記憶していく形式で表現しなければならない。又，配列などに代表される大きなデータ構造の一部の変更に対しても大量のデータ生成操作が必要となる。

これらの問題を解決するために，ストリームと呼ばれるデータ構造を導入し，履歴依存性に適応する試みあるいは，2進木のデータ構造と参照数を用いたデータ構造操作などが既に報告されている。[55],[56]。

本データ駆動図式では，履歴依存性を有する処理の記述のため，図式中に履歴を示すノード及びそれへのアクセスを示すアークを導入し，これによる非決定性を制御アークによる新たなデータ従属性の付与により回避する。従って，本データ駆動図式は，履歴依存処理の記述に関しても，その並列処理構造と同様の図的表現の了解性を保存している。更に，拡張されたデータ駆動図式によって記述されるプログラムは，履歴を示すノードへのアクセス規則を含む実行規則により，副作用のない実行が保証される。

3.3.1 履歴依存処理の記述

履歴依存性を有する処理の記述のため，図3.5に示すデータ駆動図式の構成要素を追加する。データベース(Database)は履歴を，参照(Get)，及び更新(Put)アークはその履歴へのアクセスを示す。即ち，参照あるいは更新アークを付与されたプリミティブ(History-sensitive primitive)に割当てられる関数は，いわゆる履歴依存性を有する。図3.1及び図3.5に示す要素を用いて記述されたデータ駆動図式中の各ノードの実行順序は，既に述べたように半順序である。従って，図3.6に示すように各履歴へのアクセス順序が決定されていない場合は，図3.7に示す制御構造(Shis)を用いて制御アークによる順序付けを行う。図3.8に履歴依存処理の簡単な具体例を示す。この図式中のF_0，F_1，$F\,gp_2$，$F\,gp_3'$にNAND論理ゲートを割当てると，I_1がセット，I_2がゲート，I_3がリセットの各入力，O_1，O_2がQ，$\overline{Q}$の各出力となる

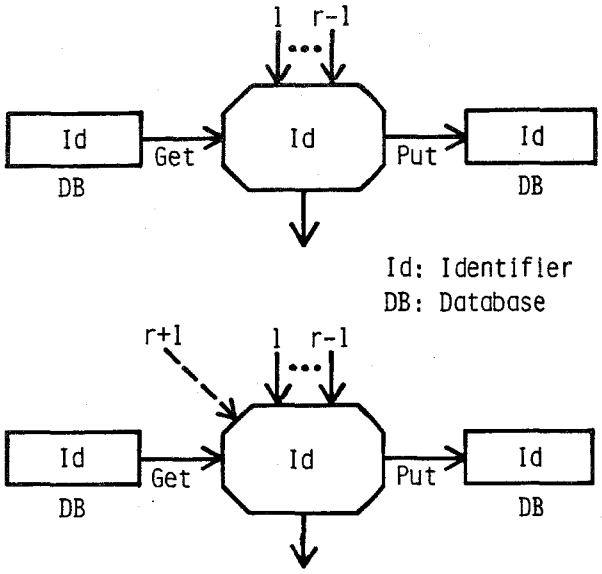


図 3.5　履歴依存性を持つプリミティブ

Fig. 3.5－History－sensitive primitive.

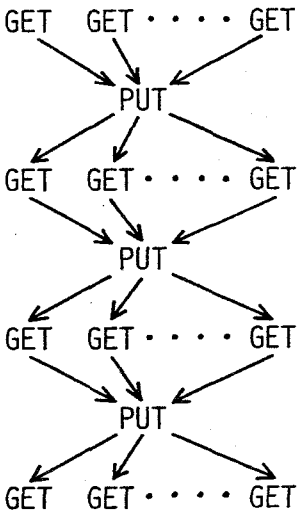


図 3.6　履歴への副作用のないアクセス順序

Fig. 3.6－Access sequence to a history without any side-effects.

一種のラッチを示すプログラムが生成される。

〔定義 3. 8〕 図 3.1 及び図 3.5 に示す構成要素を用いて，定義 3.1 に従い，且つ制御構造を Scon, Ssel, Ssel′, Srec, Srec′ 及び Shis に限定して記述され，図 3.6 のように各履歴へのアクセス順序が決定されているデータ駆動図式を S_6 とする。

〔定義 3. 9〕 S_6 の解釈 I_1 は次のようである。

①定義 3.2 に示す解釈 I_0 。

②参照アーク上のトークンの生成と消費は，すべてのデータ及び制御アークにトークンが揃った時に行われ，そのトークンの内容は履歴と同一である。

③更新アークへのトークンの生成は，出力データアークと同様に行われ，そのトークンの内容は出力データアーク上のトークンと同一である。

④更新アーク上に v （∈ V）が生成された時，v は消費され履歴はその内容に変更される。但し，Nil が生成された時，Nil は消費され履歴は保存される。

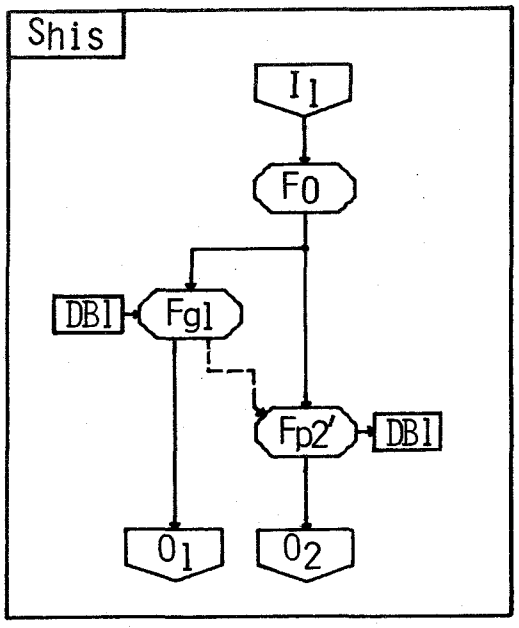


図3.7　履歴へのアクセス順序を決定する制御構造：Shis

Fig. 3.7－The construct for the access sequence to a history: Shis.

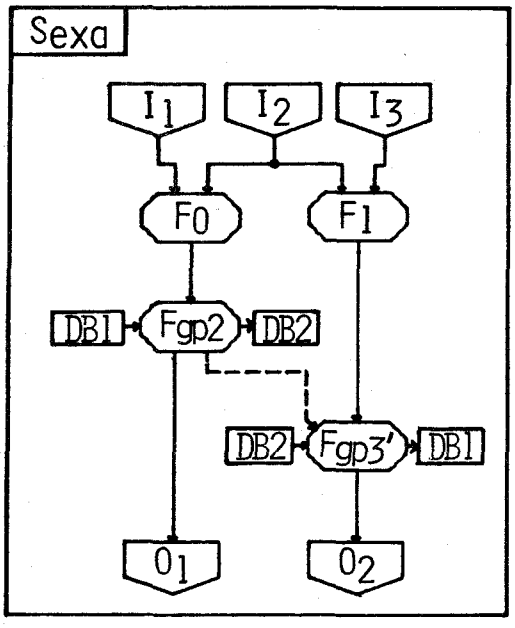


図3.8　履歴依存処理の例

Fig. 3.8－An example of history-sensitive process: Sexa.

⑤履歴へのアクセス順序を決定するために制御アークをその出力として付与されたプリミティブあるいはプロシージャは，vを生成した時Tを，Nil を生成した時Nilを，その制御アーク上に生成する。

〔定義3. 10〕データ駆動図式S_6とその解釈I_1との組〈S_6，I_1〉を履歴依存性を許すデータ駆動プログラムDP_6とする。

〔定義3. 11〕DP_6の実行は各履歴に初期値を与えられた後，定義3.4に従って実行される。

3. 3. 2　履歴依存処理における副作用の回避

DP_6は，各履歴へのアクセス順序が図3.6のように決定されているので，明らかに，定義3.11に基づいて実行される限り副作用なしである。しかし，デ

ータフロー図式には，元来，並行処理のみならずパイプライニングが可能な並列性が陽に示されている。パイプライン形の並列実行の環境では，次に定義する世代の異なるトークンの履歴へのアクセスのタイミングによって，DP_6 の実行結果に非決定性が生じる。

〔定義3.12〕 DP_6 中にパイプライン状に入力あるいは生成される複数のトークンを識別するための入力および生成順序に基づいて決定される識別子を，そのトークンの世代とする。

〔性質3.6〕 トークンの追越しのないパイプライン形の並列実行環境で，各履歴へのアクセスが次の規則に従う時，DP_6 の実行結果に非決定性は生じない。

①参照アークへのトークンの生成は，直前の更新が終了した時，許される。但し，参照から開始されるアクセスにおける最初の参照は初期値が与えられた時，許される。

②更新アーク上のトークンの消費は，常に許される。但し，更新の対象となる履歴への他の世代のトークンによるアクセスが終了していない場合，その履歴を保存するため，その更新結果を新たに生成する。

（証明） DP_6 を示すデータ駆動図式 S_6 は，定義3.8に示すように，その制御構造を限定して記述され，各ノード間のデータ従属性に関する半順序関係を保証している。従って，トークンの追越しがない時，DP_6 に履歴依存性がなければ明らかに性質3.6は成立する[55]。即ち，DP_6 の履歴依存性を有する部分（Shis） にのみ着目すれば良い。DP_6 における履歴へのアクセス順序は，①更新から開始されるアクセス，②参照から開始されるアクセス，のいずれかである。①の場合，この更新規則により履歴は各世代のトークンに対して単一に割当てられるので，DP_7 には副作用は生じない。②に関して一般性を失わないデータ駆動図式例（Shis′） を考える。〈 Shis′，I_1 〉で定義されるデータ駆動プログラムは，世代の異なる二つのトークンが入力された時，図3.9に示すデータ駆動図式 S_7 とその解釈 I_1 の組〈 S_7 ，I_1 〉で定義されるデータ駆動プログラムと同じ実行系列を持つ。即ち，S_7 はShis′を二重化

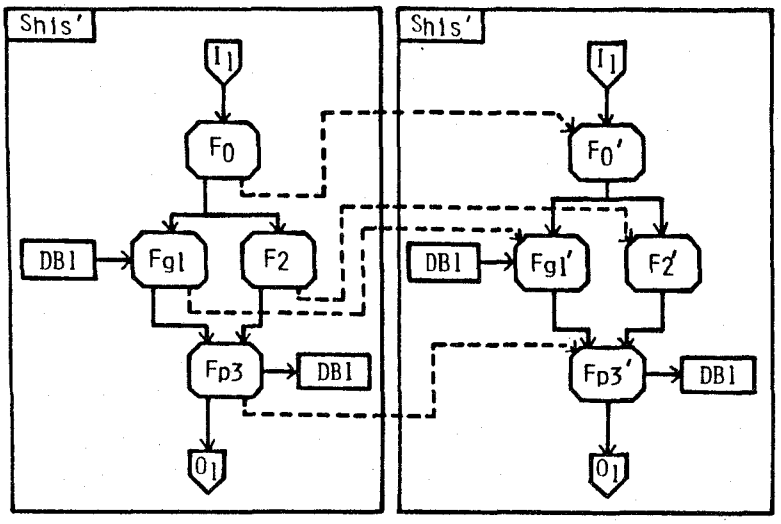


図 3.9　トークンが連続して入力された場合の
実行系列を示すデータ駆動図式例：S_7.

Fig. 3.9 −An example data−driven schema that shows execution sequences in pipelined environment : S_7.

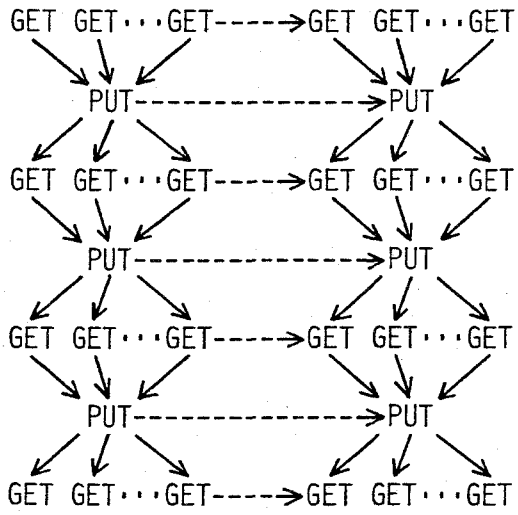


図 3.10　パイプライン環境における履歴へのアクセス順序

Fig. 3.10 − Access sequence to a history in pipelined environment

し，対応する各ノードに対してトークンの追越しを禁止する制御アークを付与して記述される。従って，図3.6に示した履歴へのアクセス順序は，パイプライン環境では図3.10に示すようになり，1世代目のトークンによる最後の更新と2世代目のトークンによる最初の参照との間にデータ従属性がないので，副作用が生じる可能性が残される。しかし，参照に関する規則は，その間の順序を保証するので，パイプライン環境における履歴へのアクセス順序は決定され，副作用は生じない。又，世代の異なる三つ以上のトークンが入力された場合も同様である。従って，DP_6 の実行結果に非決定性は生じない。

（証明終）

次に，このアクセス規則を持つ履歴の実現例を示す。

〔定義3. 13〕 S_6 に含まれる各履歴に対する参照および更新アークの数を，それぞれ参照数，更新数とする。

各履歴を図3.11(a)に示す識別子，世代，参照数および更新数をタグとして付与された記憶セルとして扱う。但し，各記憶セルの世代は，アクセスを許されるトークンの世代を示す。従って，初期値として与えられた履歴を持つ記憶セルの世代は1となる。

次の手続により，履歴の参照および更新を行う。

〔参照〕 1° 識別子，世代の一致する記憶セルが既に生成されていれば2°へ。それ以外は1°を繰返す。

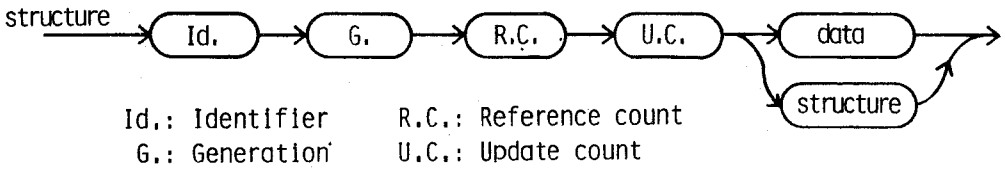
2° 参照数を1減じ，履歴を参照し，参照アーク上にトークンを生成し3°へ。

3° 参照数および更新数が共に0であれば，その記憶セルを消費して終了。それ以外は終了。

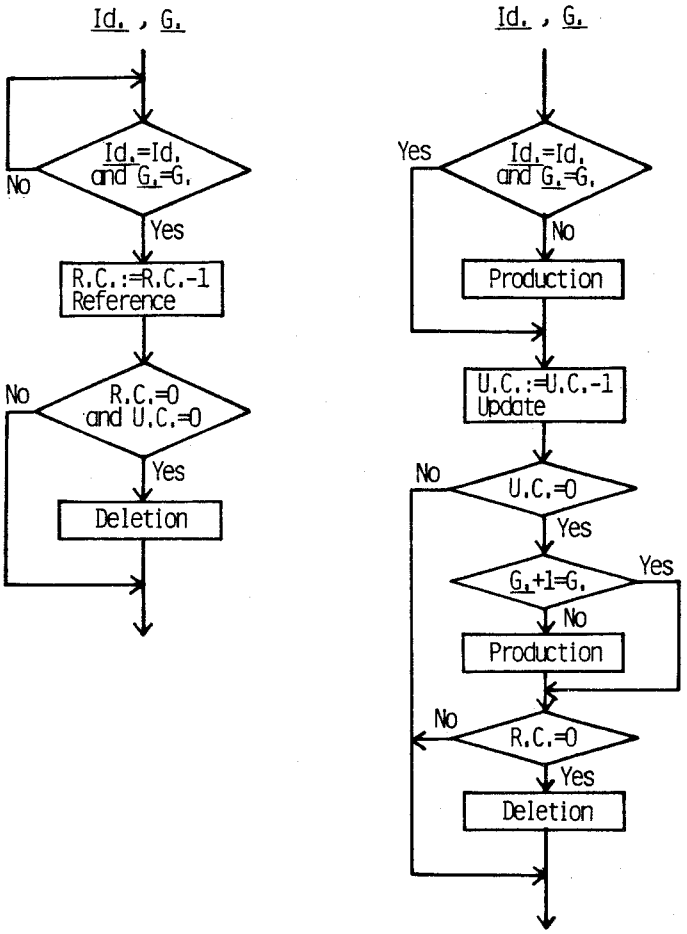
〔更新〕 1° 識別子，世代の一致する記憶セルが既に生成されている場合2°へ。それ以外は，その世代に対するタグを付与した記憶セルを生成し2°へ。

2° 更新数を1減じ，更新アーク上のトークンを消費し，履歴を更新し3°へ。

3° 更新数が0でなければ終了。更新数が0で且つ次の世代の記憶セルが既に生成されていれば4°へ。それ以外は次の世代に対するタグを付与した記憶セルを生成して，その世代の履歴を複製し4°へ。



(a) Tagged data structure.



(b) Procedures of reference / update.

図3.11　タグ付き記憶セルの一実現法

Fig. 3.11—An implementation of tagged memory—cell.

4° 参照数が0であれば，その世代の記憶セルを消費し終了。それ以外は終了。

これらの手続を図3.11(b) にフローダイアグラムの形式で示す。但し，図中のId.，G.は各トークンの，Id.，G.は各履歴の識別子，世代をそれぞれ示す。又，このアクセス規則の下では，履歴依存性による副作用の回避のために生成される記憶セル数の上限を高々，定義 3.12 で述べた世代数に抑えることができる。即ち，この記憶セルは履歴依存性を許すデータ駆動形実行環境における副作用の回避を必要最低限のコピーによって可能とするものである。

3.4 結　言

本章では，副作用のない並列処理構造を陽に記述する有力な手段として，図的表現の了解性およびデータ従属性に基づく並列性を背景に，履歴依存性にも適応できる階層的データ駆動図式を提案した。

本データ駆動図式は，処理と制御の機能を分離しノードとアークに各々割当てる仕様と，ノード間の半順序関係を保証できる制御構造を持つ。更に，本図式は，履歴依存性による副作用も回避できる実行規則を採用している。従って，記述されるプログラムはトークンの流れに関する限り，厳密にデータ駆動原理に基づいて実行され且つ副作用を生じない。

又，本図式を採用した D^3L の記述能力の確認のため，いくつかの例題を取り上げ実際に記述した結果，

①図的表記によって，高度並列処理構造の表現を含めて，階層的かつモジュラなプログラムが記述できる，

②履歴依存処理を許すデータ駆動形実行規則によって，一般に並列処理の適用分野とされている実時間制御問題の記述に対処できる基本的な能力を備えている，

などの利点を持つ反面，

①構造的に類似した処理が大量に繰返される場合の記述法が煩雑である，

②アレイ，構造体データなどの扱いができない，

などの問題点があるという結論を得ている[50]。

これらの問題への対処のため，規則的な接続構造の一般項的な記述法，並びに本章では，データ駆動図式中の履歴を単なる参照および更新が可能な記憶セルとして扱ったが，図式中の履歴として，データ構造操作機能により定義される，いわゆる抽象データ型（特にストリーム）の導入を検討している。

更に，本文中の図 3.8 にも示したように本図式はゲートレベルの論理回路の動作を比較的簡単に表現可能であることから，この図式表現を基礎にＶＬＳＩの設計援助となりうる，高度並列処理形論理回路シミュレーションシステムの構築も検討中である。

第4章　データ駆動原理の一実現法

4.1　緒　　言

本章では，D^3Lによって表現されるデータ駆動原理の一実現法として，対象とする図的プログラム構造を反映した階層的なクラスタ構成をとるシステム上で，そのデータ駆動形処理実行機能の負荷・機能分散的実行を統一的に実現できる実行制御の一方式について述べる[31]。

データ駆動原理による高度並列処理の実現には，D^3L プログラムが陽に示すアルゴリズムに基づく並列処理のみならず，実行時に動的な負荷分散を行い，入力ストリームに応じた滞留のないデータ流をシステム内に確保しなければならない。このため，対象となるプログラムの並列処理構造に適応したシステムの構成法および機能・負荷分散的実行を可能とする制御方式が必要となる。

本方式では，対象とする階層的な図的プログラムのデータ駆動形処理実行機能を，入出力処理，関数的処理，履歴依存処理ならびに発火制御の四基本機能の繰返し構造で構成し，これらの基本機能の機能および負荷分散的な実行を可能とする複数の資源割当機構により結合して，極めて高度の同時・並行処理を統一的な方法で実現している。即ち，本システムでは，図的プログラムを任意の複数の階層に分割し，特定の階層を同一の資源割当機構に結合された基本機能群（クラスタ）に対応させる方法をとる。又，ある階層のクラスタ内の各基本機能は，必要に応じて，更に下位のクラスタの入れ子構造とすることを許している。従って，本システムでは，対象とする図的プログラムの幾何学的構造を反映した階層的クラスタ構成上での基本機能の負荷ならびに機能分散的実行を自由に実現できる。

更に本システムは，次のような優れた特性を示す。

①資源の適正な分配が自動的に行われ，実行時の動的な並列処理が自然に実行される。

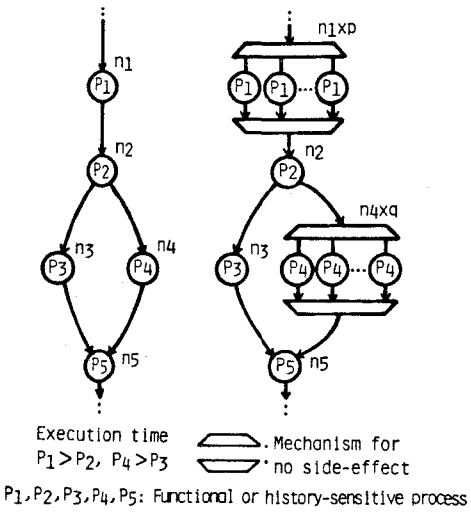
②システムの規模および構成にかかわらず，統一的な実行制御が可能で，モジュラな拡張性を持つ。

4．2　データ駆動原理による高度並列処理

本節では，まず D^3L プログラムの効果的な実行制御には入力ストリームに応じて，滞留のないデータ流をシステム内に確保できる負荷・機能分散方式が必要であることにふれる。次に，データ駆動原理に基づくシステムの動作をモデルにより解析し，その負荷・機能分散が行える前提条件について考察する。

4．2．1　実行時の動的負荷分散

D^3L プログラムの記述形式であるデータ駆動図式は，処理内容を示すノード，データと制御の推移を示すアークから構成される有向グラフである。即ち，この図式はデータ従属性に基づく並列処理構造（同時並行，パイプライニング）を陽に示している。従って，データが次々とストリーム形に入力される実行環境下で，D^3L プログラムの持つ並列性が最大限に引出される。しかし，図4.1(a)のように，各ノード $n_1 \cdots n_5$ に割当てられる関数的あるいは履歴依存処理 $P_1 \cdots P_5$ の実行時間に差がある場合，P_2 の駆動間隔は P_1 の実行時間に



(a) static parallelism.　(b) dynamic parallelism.

図 4.1　動的に効果的なデータ駆動形実行

Fig.4.1—Dynamically efficient data—driven execution.

よって，又，P_5 の駆動間隔はP_4 の実行時間によって制限されるため，データの滞留が生じる。これの解消には，図4.1(b)のように，データの追越しによる副作用の回避を保証した上で，P_1 及びP_4 の負荷分散的な同時並列実行が必要となる[57],[58]。即ち，効果的なデータ駆動形実行制御の実現には，対象となる D^3L プログラムの並列処理構造およびその規模に適応できるシステムの構成法のみならず，入力ストリームに応じたデータ流をシステム内に確保するための柔軟な負荷・機能分散方式を採用しなければならない[57]。

4.2.2 データ駆動システムの動作モデル

D^3L によって表現されるデータ駆動原理に基づき動作するシステムは，図4.2のようにモデル化できる。即ち，このシステムは，図式中のアークの接続

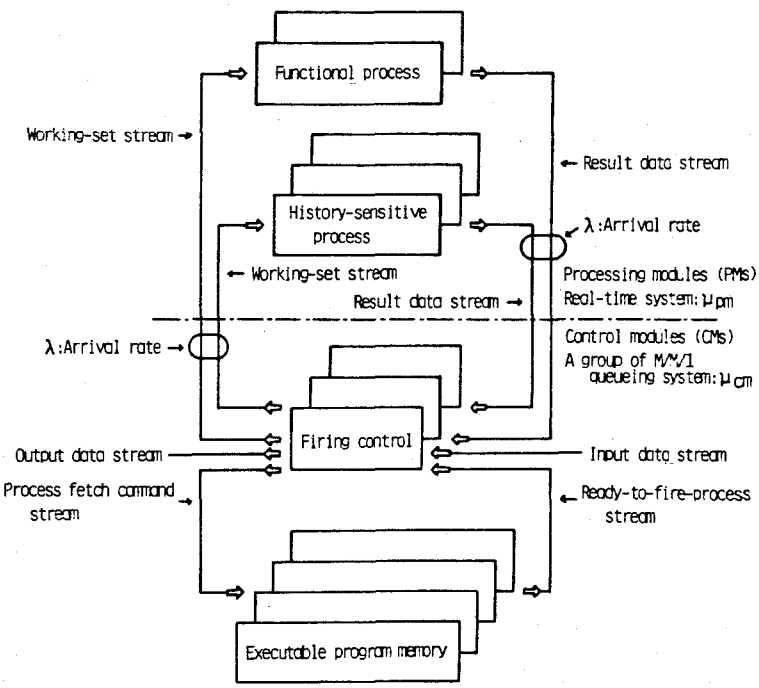


図4.2　データ駆動原理の機能構成

Fig.4.2－Functional organization in the data-driven principle.

構造が示すデータ従属性に従って，図式中のノードに割当てられた各処理の並列実行の駆動を行う，実行形式プログラム記憶(Executable program memory)と発火制御(Firing control)からなる制御モジュール群(Control modules; CMs)と，CMsによって駆動される関数的処理(Functional process)並びに履歴依存処理(History-sensitive process) を並列実行する，処理モジュール群 (Process modules ; PMs) 及びそれらを結合する通信網から構成される。更にこのシステムは，入力データストリーム (Input data stream)によって起動され，各モジュールが発火制御部を中心にして，データ駆動形に実行制御され，図4.2に示すようなデータストリームの変換を行うことにより，処理結果として出力データストリーム (Output data stream) を生成する。ここで，CMs 中の一つのモジュールCMに着目すると，CMはPMs を駆動するM／M／1の待ち行列システム (M／M／1 queueing system), PMs は並列処理を実行する待ちのないシステム (Real-time system)として動作すると考えられる。PMs 内では 1 in 1 out の原則に基づきデータの消滅は起こらないので，CM及びPMs へのデータの到着率 (Arrival rate)は等しく，これを λ とし，μ_{cm} ，μ_{pm} をCM，PMs のサービス率(Service rate)とする。λ はCMとPMs で構成されるシステム内のデータ流P，即ちストリーム処理能力を示すと考えられる。Pは，CM及びPMs 中のデータ数の合計に等しいので次式が成立する。

$$\lambda／(\mu_{cm}-\lambda)+\lambda／\mu_{pm}=P$$

$$\lambda^2-(P\cdot\mu_{pm}+\mu_{pm}+\mu_{cm})\lambda+P\cdot\mu_{pm}\cdot\mu_{cm}=0$$

$B=P\cdot\mu_{pm}+\mu_{pm}+\mu_{cm}$，$C=P\cdot\mu_{pm}\cdot\mu_{cm}$ として，λ ($<\mu_{cm}$，μ_{pm}) について解くと，

$$\lambda=\frac{1}{2}\cdot\{B-(B^2-4C)^{\frac{1}{2}}\}$$

$$=2C／\{B+(B^2-4C)^{\frac{1}{2}}\}$$

ここで，与えられたシステム資源を余すことなく有効に活用して，D^3L プログラムが並列実行される理想的な場合には，$P\cdot\mu_{pm}\fallingdotseq\mu_{cm}$ の平衡条件が成立する。又，$P>>1$ であるから，$\mu_{pm}<<\mu_{cm}$ となる。故に，$\lambda\fallingdotseq P\cdot\mu_{pm}$ 。

PはCMに割当てられる D^3L プログラムの並列処理構造，即ち，平均的に同時に並列実行可能な処理数により決定される。又，μ_{pm} は PMs を構成するモジュール数 N_{pm} に比例して大きくなる。故に，システムに入力されるストリームの増大に対してCMs のモジュール数 N_{cm} の追加により対処できる機能・負荷分散方式を採用すれば，システム資源（ N_{pm} 及び N_{cm} ）の投入に比例したストリーム処理能力の向上，即ち高度並列処理が実現される。従って，その前提条件として，① $\mu_{pm} << \mu_{cm}$ を保証するための並列実行されるタスクレベルの設定，② $P \leq (\mu_{cm} / \mu_{pm})$ を考慮したCMs への D^3L プログラムの分割割当，が必要となる。

4．3　データ駆動形実行制御の一方式

本節では，データ駆動原理の一実現法として，D^3L プログラムの階層的なブロック構造を反映したクラスタ構成上で，そのデータ駆動形実行機能の負荷ならびに機能分散を統一的に扱える方式について述べる。まず D^3L プログラムのデータ駆動形実行に必要となる基本機能群を示す。次に，各基本機能の分割割当とシステム構造そのものに内在された資源割当機構によって可能となる負荷・機能分散方式について述べる。

4．3．1　D^3L プログラムのデータ駆動形実行機能

D^3L プログラムは図4.3のような階層的なブロック（Block)構造を持つ。各ブロックはデータ駆動副図式によって記述される副プログラムを示す。本データ駆動図式は，入出力表明（ Source，Sink ），関数的処理（ Primitive ），手続（ Procedure ），条件判断（ Predicate），履歴（ Database ）を示すノード及びデータ従属性(Data，Control)，コピー(Copy)，マージ（Merge)，履歴の参照・更新（ GET，PUT ）を示すアークから構成される。又，副プログラムの制御構造には単なる連接だけでなく，選択（Conditional），再帰（Recursive），並びに履歴依存（History-sensitive）が許されている。即ち，D^3L プログラムの記述形式は履歴依存性を含む任意のアルゴリズムに内

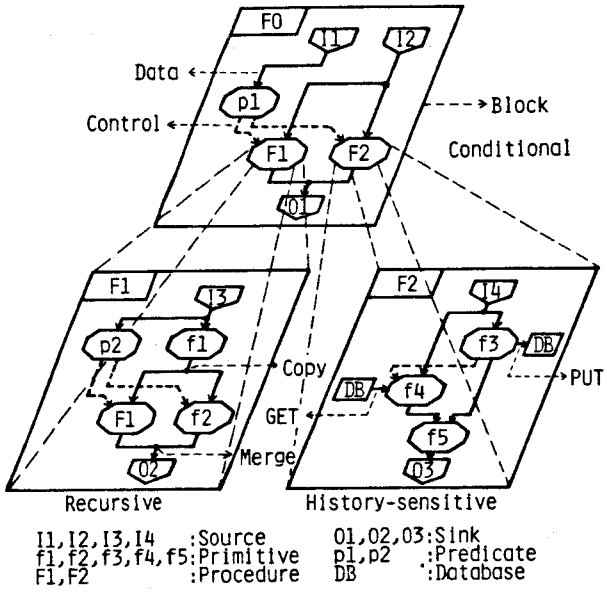


図4.3　D^3L プログラムの階層的ブロック構造とその制御構造

Fig.4.3 －Hierarchical block structure and constructs in the D^3L program.

在する並列処理構造（同時並行処理，パイプライニング）を陽に示している[30]。

このような D^3L プログラムのデータ駆動形実行のための基本機能群の設定には多くの選択の余地があるが，対象とする D^3L プログラムの並列処理構造およびその規模に対処できる拡張性が必須となる。ここでは，その第一歩として D^3L に採用したデータ駆動図式の記述要素から抽出される以下の機能群を採用した。

①入出力処理…ソースノード，シンクノード

②関数的処理…プリミティブノード

③履歴依存処理…データベースノード，参照／更新アーク

④発火制御…プレディケートノード，データ／制御／コピー／マージアーク

これらの基本機能群によってクラスタを構成し，更に，プロシージャノードによって示されるブロック呼出の機能を，クラスタの相互接続機能により実現する。この機能は，D^3L の解釈から推測できるように，①及び②の機能を有

するが，各クラスタ内では①或いは②のいずれか一方として動作する。即ち，D^3L の解釈に基づいた基本機能の設定により，D^3L プログラムの並列処理構造およびその規模に適応できる任意のクラスタ構成がこれら四基本機能の繰返し構造で実現できる。

4.3.2 データ駆動形実行機能の負荷・機能分散方式

D^3Lプログラムの実行制御の高効率化の問題は，本質的には実行可能な処理を高速に検出する手段の工夫に帰着する。この問題のとらえかたは種々あるが，まず第一に対となるデータの組をなるべく簡単な機構で高速に発見する問題と考えられる。これらを考慮して連想記憶を用いる手法などが提案・検討されている。無限の資源の下で D^3L プログラムを実行する場合には，この機構だけで充分である。しかし，資源制約下でデータ流を最大にするためには，簡単なプログラムでも，その適当な分割配置による最適化あるいは実行順序の制御，即ち資源の配置・割当という問題が生じる。特に，履歴依存処理を有限の資源つまり記憶量の制限の下で実行する場合には，デッドロックが生じる可能性がある。即ち，D^3L プログラムの効果的な実行制御には，その階層構造や制御構造に基づく局所性を考慮した副プログラムの分割配置，各基本機能の分割割当と共に，副プログラムの並列処理構造に適応した資源割当方式によって可能となる機能・負荷分散的実行を実現しなければならない。従って，本方式では図4.4に示すように，対象とする D^3L プログラムの幾何学的接続構造に適応できる階層的なクラスタ構成を採用すると共に，クラスタ間およびクラスタ内のデータ駆動形処理実行機能の負荷・機能分散を統一的な方法で実現している。即ち本方式は，階層的なシステムにおける D^3L プログラムのデータ駆動形処理実行のための基本機能群の結合を単なる通信網でなく，資源割当機構（Resource allocation mechanism ; RM）[59)-(63)]を用いて行い，これらの基本機能の機能・負荷分散的実行を可能とするため，データ駆動形の通信をすべてRMを介した次のようなパケット交換により実現する。

入出力サブシステム（I／O Subsystem ; SS）群はそれぞれ外界からの

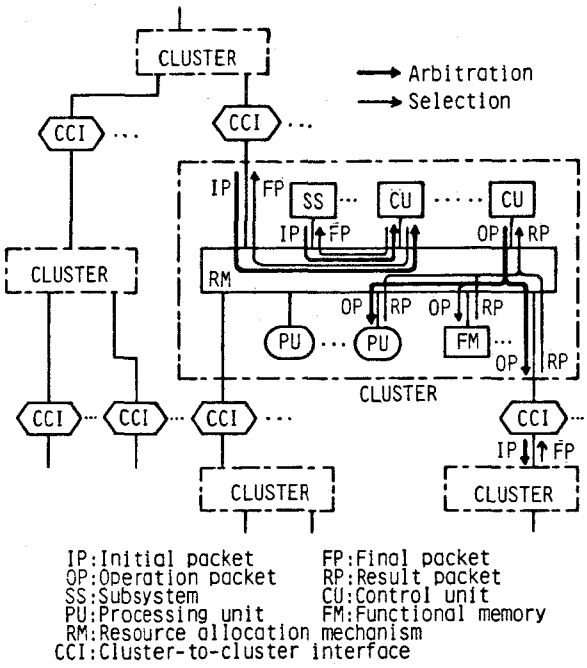


図 4.4　　データ駆動形実行制御の一方式の機能構成

Fig. 4.4—Functional organization in a data-driven execution control.

処理要求により駆動され，初期パケット（Initial packet ；ＩＰ）を生成し，制御ユニット（Control unit ；ＣＵ）群へ送信する。ＣＵは，ＩＰの受信により駆動され，副プログラムのデータ従属性に基づいて，機能メモリ（Functional memory ；ＦＭ）或いは処理ユニット（Processing unit ；ＰＵ）を駆動するため，オペレーションパケット（Operation packet ；ＯＰ）の送信，レザルトパケット（Result packet ；ＲＰ）の受信を繰返し，処理終了後，最終パケット（Final packet ；ＦＰ）をＳＳ群に送信する。ＦＭはＯＰの受信により駆動され，副作用を回避できる規則[30]に従って割当てられた履歴の参照・更新を実行し，ＲＰを送信する。ＰＵはＯＰの受信によって駆動され，関数的処理を実行した後，ＲＰを送信する。ここで，ＲＭはパケットの送信要求の調停だけでなく，各パケット中に示される処理内容に基づき，受信先の決定を行える選択

的結合機能を持つため，次のような各基本機能の割当により，それらの負荷・機能分散的実行が可能となる動作モードがクラスタ内に実現される。

即ち，ＦＭ群，ＰＵ群に履歴，関数的処理を分割配置し，ＯＰをその割当てられた機能に基づいて，選択（Selection）的に受信させれば，機能分散的実行が，又，履歴依存性を持たないＰＵ群では，機能割当に冗長性を付与し，同じ関数的処理機能を持つＰＵ間でＯＰを裁定（Arbitration）的に受信させれば負荷分散的実行が実現される。更に，ＣＵ群に副プログラムのデータ従属性を斉一に割当て，ＩＰの裁定的な受信により，データのいわゆる世代に関して負荷分散させる。この方式では，4.2.1に述べた同時並列実行が，ＰＵ群およびＣＵ群における負荷・機能分散により自然に実現されるだけでなく，4.2.2に述べた入力ストリームの増大への対処もＰＵ及びＣＵの追加により行える拡張性をシステムに付与できる。更に本方式は，図4.4に示すパケット変換機能を持つクラスタ間インタフェース(Cluster-to-cluster interface ; ＣＣＩ）を用いて，上述のクラスタの相互接続を可能としている。ＣＣＩはＳＳ及びＰＵの機能を持つが，各クラスタ内ではＳＳ或いはＰＵのどちらかとして動作するので，本方式では，任意のクラスタ構成の実行制御がＳＳ，ＣＵ，ＦＭ及びＰＵ群からなる繰返し構造で実現される。

従って，本方式では，データ駆動形処理実行のための各基本機能の負荷・機能分散的実行を可能とする資源の適正な分配がＲＭの選択的結合機能により自然に行われると同時に，対象とする D^3L プログラム構造および入力ストリームに応じたクラスタ構成が各基本機能のモジュラな増設によって実現できる。

4.4 結　言

本章では，D^3L によって表現されるデータ駆動原理の一実現法として，階層的なクラスタ構成をとるシステム上で，データ駆動形処理実行機能の負荷・機能分散を統一的に実現できる方式を述べた。

本方式では，D^3L のデータ駆動形処理実行のための基本機能として，その記述形式，即ちデータ駆動図式から抽出される，入出力処理，関数的処理，履

歴依存処理，並びに発火制御機能を設定し，これらの機能・負荷分散的な実行を可能とする動作モードをシステム構造そのものに内在される資源割当機構によって実現している。従って，対象とする D^3L ，即ち中間言語プログラムの構造・規模に適したクラスタ構成および適正な処理レベルの設定による，高度並列処理の実現可能性を基本的に検討できると共に，ある特定の問題向きの専用的システムを構成する際の基本機能構成などの設計指針を提供することが期待される。残された問題としては，効果的な発火制御機構の検討および対象とする問題の構造に応じた上述の四基本機能の分割・併合割当法などがある。

第５章　高度並列処理実現手法に関する実験的検討

5.1　緒　　言

本章では，共通バス形式のマルチプロセッサ構成をとる実験システム[29],[31]を用いた，D^3L による高度並列処理実現手法に関する基本的な実験的検討結果を述べる。

本実験システムの構成は，ごく一般的な共通バス結合マルチプロセッサシステムと同様である。しかし，必要に応じて二つのバス系にまたがるプロセッサを用いて，複数のバス系の相互接続によるクラスタ構成を採用できると共に，バス系に導入された二組の二線式非同期リングアービタによって実現される各プロセッサ間の裁定あるいは選択的結合の機能が与えられている。そこで，本システム上では，対象とする図的プログラムを任意の複数の階層に分割し，特定の階層を同一のバス系に結ばれた基本機能群（クラスタ）に対応させる方法をとる。又，既に述べたような，ある階層のクラスタ内の各基本機能を更に下位のクラスタの入れ子構造とすることを，バス系の相互接続によって可能としている。従って，本システムでは，対象とする図的プログラムの幾何学的構造を反映した，複数の階層的クラスタへの負荷ならびに機能分散を自由に実現できる。共通バス結合形マルチプロセッサシステムは一般に，バス上のデータ転送能力が隘路となって，大規模なシステムの物理的構成手法として不向きであるとされているが，本実験システムは大規模構成においても，この問題を軽減できる特徴を持っている。

又，本実験システムは，システム動作モードの柔軟性により，対象とする図的プログラムの実行機能のみならず，図的プログラムを含むシステム自身の開発援助機能あるいはシステムの動作状況観測などの諸機能をシステム内に統合的に実現できることも特徴としている。即ち，このバス系は，一種のパケット交換網として動作するので，本実験システムでは，4.3.1 に述べた D^3L プログラムのデータ駆動形処理実行のための各基本機能をプロセッサ上の簡単な構造のソフトウェアにより実現できる[64]。更に，本実験システムには，対象とす

るD^{3}Lプログラムの開発援助のためのグラフィックエディタ，トランスレータ，マッピングローダなどからなるD^{3}Lの処理系[65]と共に，システム自身の開発援助あるいはその動作観測などの諸機能を果たすハードウェアモニタインタフェース(Hardware monitor interface ; HMI)[66]が付与されている。例えば，システムの一部にバス上のデータ流の観測機能を付与し，処理資源をモジュラに増設すれば，自己増殖的にシステムの物理的構成規模とその機能とが拡張される。

以下本章ではまず，実験システムのハードウェア構造とその機能，並びに本システム上に実現されたD^{3}Lの一実行形式とその処理系について簡単に述べる。次に，本システムを用いた実験方法にふれた後，適正なプログラムの分割配置・タスクレベルを設定した場合，本方式がシステム資源の投入にほぼ比例したストリーム処理能力の向上，即ち高度並列処理を実現しうることを示す。更に，対象とするD^{3}Lプログラムの幾何学的接続構造に基づき，本方式における各基本機能を階層的なクラスタ構成上へ分割配置するための指針を半定量的に提案している。

5.2 実験システムとその特徴

本節ではまず，単位共通バス系(クラスタ)内の機能・負荷分散方式の原理を簡単に述べる。続いて，図的プログラムのデータ駆動形実行のための基本機能を各プロセッサ上の簡単な構造のソフトウェアにより実現する手法を示すと共に，D^{3}Lプログラムの処理系について簡単にふれる。

5.2.1 ハードウェア構造とその機能

本システムは，図5.1のようにマイクロプロセッサZ－80を基本とするシングルボードコンピュータ(Single board computer ; SBC)をバスインタフェース(Bus interface ; BI)及び二組のアービタを持つバスバッファ(Bus buffer ; BB)を介して共通バス(Common-bus)に接続する構成を基本としている。二つのアービタで形成される二組のループ(Arbiter loops)[61]，即ちエミッタ

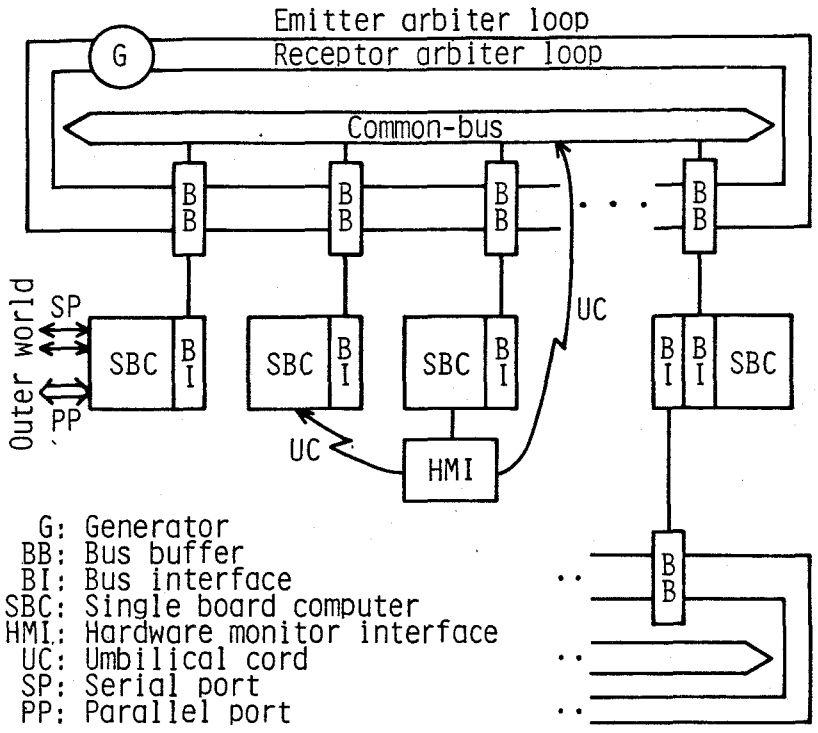


図5.1　実験システムのハードウェア構造

Fig. 5.1 – Hardware structure in the experimental system.

(Emitter) 及びレセプタ (Receptor) は，本システムにおける 資源割当機構 (Resource allocation mechanism ; RM) として，次のように動作する。

システム内の資源割当に能動的なＳＢＣ群はそれぞれ，エミッタによってバス使用権を与えられ，レセプタを起動し，選択符号によって，データ転送対象となる機能をバス上に示す。このとき，レセプタに接続されたＳＢＣ群，即ち資源割当に受動的なＳＢＣ群に選択符号が一致し，且つ空き状態のものがあれば，その最初のものが選択される。この選択符号は，ＳＢＣの物理的位置でなく，その機能を示す識別コード (Identification code) を用いている。又，ＢＩは識別コードのマスク機能を持ち，特定の機能識別コードを持つＳＢＣ群を一括して指定することも可能である。従って，特定のＳＢＣにその機能に固

有の識別コードを与えれば，機能分散形の選択が，複数のＳＢＣに同一（グループ）の識別コードを割当てると負荷分散形の裁定がそれぞれ可能となる。これらの機能選択とそれに続くデータのブロック転送はＢＩのハードウェア機能で行われるので，ＢＩ群と共通バス系とが一体となって，一種のパケット交換機能が実現されている[29]。

あるＳＢＣが二つのバス系（クラスタ）にまたがって接続されるとき，このＳＢＣはクラスタ間インタフェース（Cluster-to-cluster interface；CCI）として動作可能である。このバス間接続機能は既に 4.3.2 に述べたように，対象となる問題の構造に適した階層的システム構成に用いられる。

又，本実験システムのように分散的かつ非同期な動作を基本とするシステムの性能測定には，対象システムの動作に影響を与えることなく，複数の観測点についてシステム動作を確実に把握する道具が不可欠である。同時に，本システムは実験システムであるために，システムの開発あるいは変更がたびたび要求され，効果的な開発ツールの必要性も極めて高い。これらの要求を同時に統合的に満足するために，本システムにはＨＭＩが導入されている。

図 5.1 に示すＨＭＩは，いわゆるＩＣＥ機能を持ち，モニタ上のメモリからデータ流を供給し或いは対象とする観測点のデータをメモリに収集する。従って，対象とするバス上のパケット流の観測と制御によって性能評価機能を，或いはシステム内のＳＢＣの動作の監視と制御によってシステム開発機能をそれぞれ発揮する。例えば，システムのモジュラな繰返し構造を活用して，あらかじめ正常な動作を確認したＳＢＣに，ＨＭＩを介して，検査対象となるＳＢＣを接続し，ＩＣＥ機能を利用してその動作を確認する。この過程を次々に反復して，正常動作の確認されたＳＢＣを順次増加すれば，自己増殖的にシステムの物理的構成規模と機能とが拡張できる[29],[66]。

5.2.2　D^3L の一実行形式とその処理系

実験システム上でのデータ駆動形実行制御には，4.2.1 及び 4.2.2 に述べた D^3L のデータ駆動形実行のための各基本機能，並びにそれらの負荷・機能分

散的実行を可能とする資源割当方式を実現しなければならない。本実験システム上では，各基本機能を簡単な構造のソフトウェアによって実現し，クラスタ内のＳＢＣ群にそれぞれ専用的に配置している。従って，本方式では各ＳＢＣへの以下のような識別コードの割当を行えば，基本機能群の負荷・機能分散的駆動がＲＭの柔軟な資源分配機能とその動作モードによって可能となっている。

処理ユニット（Processing unit ；ＰＵ）は，実行可能な処理機能に対する識別コードを持ち，オペレーションパケット（Operation packet ；ＯＰ）を裁定（Arbitration）的に受信し，関数的処理を実行した後，レザルトパケット（Result packet ；ＲＰ）を送信する。即ち，ＰＵ群は履歴依存性を持たず，システムの資源割当に関して受動的である。このためＰＵ相互のほぼ完全な独立性の実現と共に，システムからのＰＵの独立性の付与が可能となり，ＰＵ群への機能割当に十分な冗長性を与えれば，機能および負荷分散が自然に実現される。又，機能メモリ（Functional memory ；ＦＭ）は，割当てられた履歴に対する識別コードを持ち，ＯＰを選択的に受信し，3.3.2に述べた副作用の生じないアクセス規則に従って，履歴の参照・更新を行った後，ＲＰを送信する。

制御ユニット（Control unit ；ＣＵ）は，入出力サブシステム（Ｉ／Ｏ Sub system ；ＳＳ）からの初期パケット（Initial packet ；ＩＰ）によって起動され，このクラスタに割当てられたプログラムに対応する実行形式，即ち，D^3Ｌ副プログラムの接続構造を示すテーブルに基づき，ＯＰの送信，ＲＰの受信を繰返し，処理終了後，最終パケット（Final packet ；ＦＰ）をＳＳに送信する。従って，ＳＳ及びＣＵは，ＦＰ，ＲＰを選択（Selection）的に受信するため，そのＳＢＣに固有の識別コードを持つ。ここで，同一クラスタに属するＣＵ群にその副プログラムに対するテーブルを斉一に割当て，ＢＩの識別コードマスク機能を用いてＩＰを裁定的に受信させることにより，データのいわゆる世代（Generation）に関して負荷分散させる。この方式では，実行時の負荷分散的な同時並列実行が，ＰＵ群およびＣＵ群における負荷・機能分散によって自然に実現される。同時に，入力ストリームの増大に対してもＰＵ及びＣＵのモジュラな増設により対処できる拡張性がシステムに付与される。

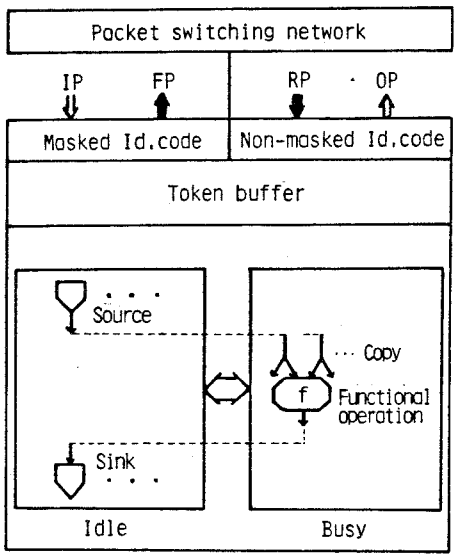
更に本方式は，図4.4に示したパケット変換動作を行うＣＣＩを用いて，上述のクラスタの相互接続を可能としている。ＣＣＩはＳＳ及びＰＵの機能を持つため，二種類の識別コードが与えられる。しかし，各クラスタ内ではＳＳ或いはＰＵのどちらかとして動作するので，本方式では，任意のクラスタ構成における実行制御がＳＳ群，ＣＵ群，ＰＵ群およびＦＭ群からなる繰返し構造で実現される。従って，本方式では，データ駆動形処理実行機能の負荷・機能分散のための資源の適正な分配がＲＭのパケット交換機能により統一的な方法で行われると共に，対象とするデータ駆動プログラム構造および入力ストリームに適したクラスタ構成が各基本機能のモジュラな増設によって実現できる。

各ＳＢＣ上の実行形式では，ＳＳの機能をＩＰの送信およびＦＰの受信プログラムとして実現している。又，ＰＵ，ＦＭはその識別コードによりＯＰを裁定あるいは選択的に受信した後，OP中の機能コード(Function code；F. code)で指定された処理を実行するため，複数の関数的処理，履歴の割当が可能である。図5.2(a)及び(b)にＣＵ，ＦＭの構造を，図5.2(c)～(n)に各パケット及びテーブルの構成を示す。但し，この実行形式[64]では各ノードに割当てられる処理を２入力，２出力までとしている。ここで，パケット中のデスティネーション(Destination；ＤＥＳＴ)はその応答となるパケット(ＩＰではＦＰ，ＯＰではＲＰ)の行先を，テーブル中のＤＥＳＴは後続するノードを指示する。又，各テーブルは８バイト，各パケットは16バイトの固定長で実現されている。

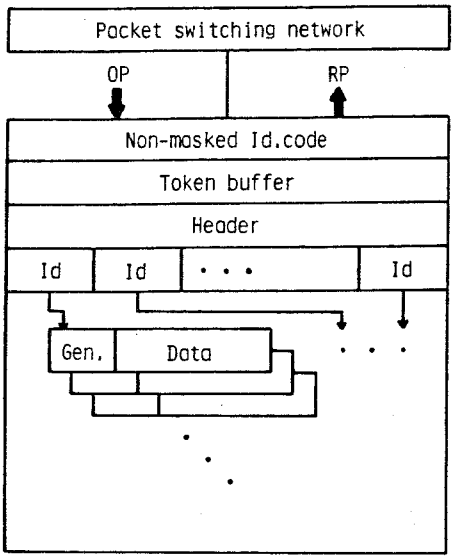
ＣＵ，ＦＭは次の手続に従って，発火制御，参照・更新をそれぞれ行う。

〔発火制御〕1° 一定時間以内にパケットを受信しなければ7°へ。ＩＰを受信した時，フラグが未使用を示すソーステーブル(Source table；ＳＴ)のヘッダエリア(Header area)にＦＰのヘッダを格納し2°へ。ＲＰを受信した時2°へ。

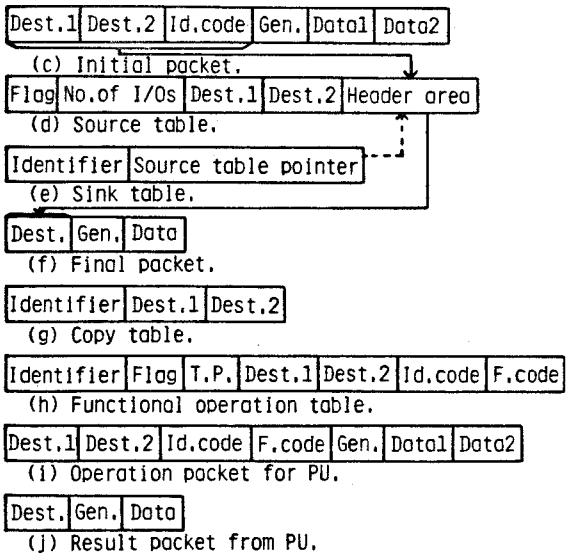
2° ＤＥＳＴの指定が，コピーテーブル(Copy table)であればそれを実行し再び2°へ。シンクテーブル(Sink table)であれば3°へ。関数的オペレ



(a) Control unit (CU) structure.



(b) Functional memory (FM) structure.



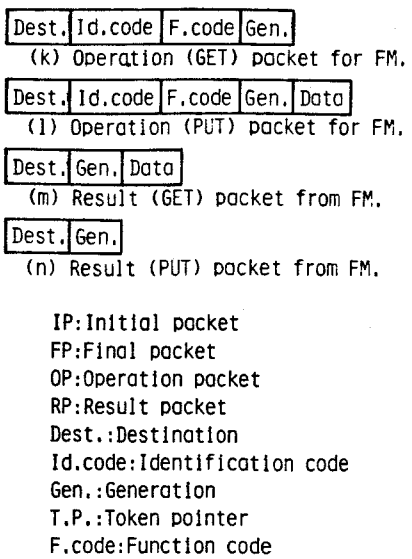


図 5.2　D^3L プログラムの一実行形式

Fig. 5.2 — An executable form for the D^3L program.

ーションテーブル(Functional operation table ; ＦＯＴ)であれば4° へ。

3° ＳＴを参照し最終パケットを生成する。もし，それが最終出力であればＳＴを解放し5° へ。それ以外は5° へ。

4° ＦＯＴの入力が揃っていればＯＰを生成し5° へ。それ以外は6° へ。

5° パケットを送出。成功すれば6° へ。失敗した時，そのパケットをキューに格納し6° へ。

6° 受信したパケットに他のＤＥＳＴがあれば2° へ。それ以外は7° へ。

7° キューに格納されている全パケットを送出。成功したパケットをキューから取除き8°へ。

8° 未使用のＳＴがあれば，所定のマスクパタンを設定し1° へ。それ以外は，マスクパタンを解除した状態で1°へ。

【参照・更新】 1° 一定時間コマンドパケット待ち。受信すれば2° へ。それ以外は7° へ。

2° コマンドがＧＥＴなら3° へ。ＰＵＴなら5° へ。

3° 識別子と世代の一致する記憶セルが既に生成されていれば4° へ。それ以外はコマンドをキューに格納し1° へ。

4° 記憶セルを参照しレザルトパケットを生成し，それを送信する。更に，それが最後のＧＥＴであればフラグを設定し6° へ。

5° 識別子と世代の一致する記憶セルを更新し，更新完了のレザルトパケットを送信する。更に，それが最後のＰＵＴであれば，次の世代の記憶セルを生成すると共にフラグを設定し6° へ。

6° フラグを参照し，最後のＧＥＴ及びＰＵＴが完了していれば，その識別子及び世代を付与された記憶セルを解放し7°へ。

7° キューを参照し，コマンドがあれば2° へ。それ以外は1° へ。

又，実験システム上には，この実行形式のサポートのため次のような機能を持つ D^3L の処理系が実現されている[28),(29),(65)]。

①グラフィックエディタ（Graphic editor ; ＥＤ）

D^3L の言語仕様に基づくブロック単位の階層的な図的プログラムの会話的開発・編集，そのハードコピー並びにライブラリ登録を行う。同時に，その副プログラムのシンタックスをチェックする。

②データ駆動形シミュレータ／バリデータ

（Data−driven simulator ／ validator ; Ｓ／Ｖ）

Ｓ／Ｖには，現在，逐次実行版と並列実行版が実現されている。

逐次形Ｓ／Ｖは，D^3L プログラム中の各処理のシミュレーション実行を，

ⅰ）Breadth-first-search（ B.F.S.），

ⅱ）Depth-first-search（ D.F.S.）及び

ⅲ）オペレータの指定による任意の順序，で行えると共に，

ⅳ）履歴へのアクセス順序が図 3.8 に示したように，更新に関して全順序・参照に関して半順序に決定されていない場合に生じる非決定性を検出できる。

又，並列形Ｓ／Ｖは，世代の異なるトークン流をグラフィック表示し，データ駆動形実行環境における，

ⅰ）ボトルネックの検出，

ⅱ）副作用の回避，

などの機能を有している。

③トランスレータ（Translator ; ＴＲＮ）

指定された複数の副プログラムの結合，未定義プリミティブ並びに再帰呼出の検出，実行形式への変換を行う。又，プログラムのシンタックス／セマンティクスも同時にチェックする。

④マッピングローダ（Mapping loader ; ＬＯＡＤ）

実行形式にデータ駆動形実行に必要な制御構造を付与すると共に，それらを対象とする実験システムの物理的構成上に分散配置する。

⑤ハードウェアモニタリングシステム（Hardware monitoring system ; ＨＭＳ）

指定されたＳＢＣ群ならびにバス系の動作状況を監視する。後述する本実行制御方式に関する実験では，主としてパケット流の観測手段として用いられて

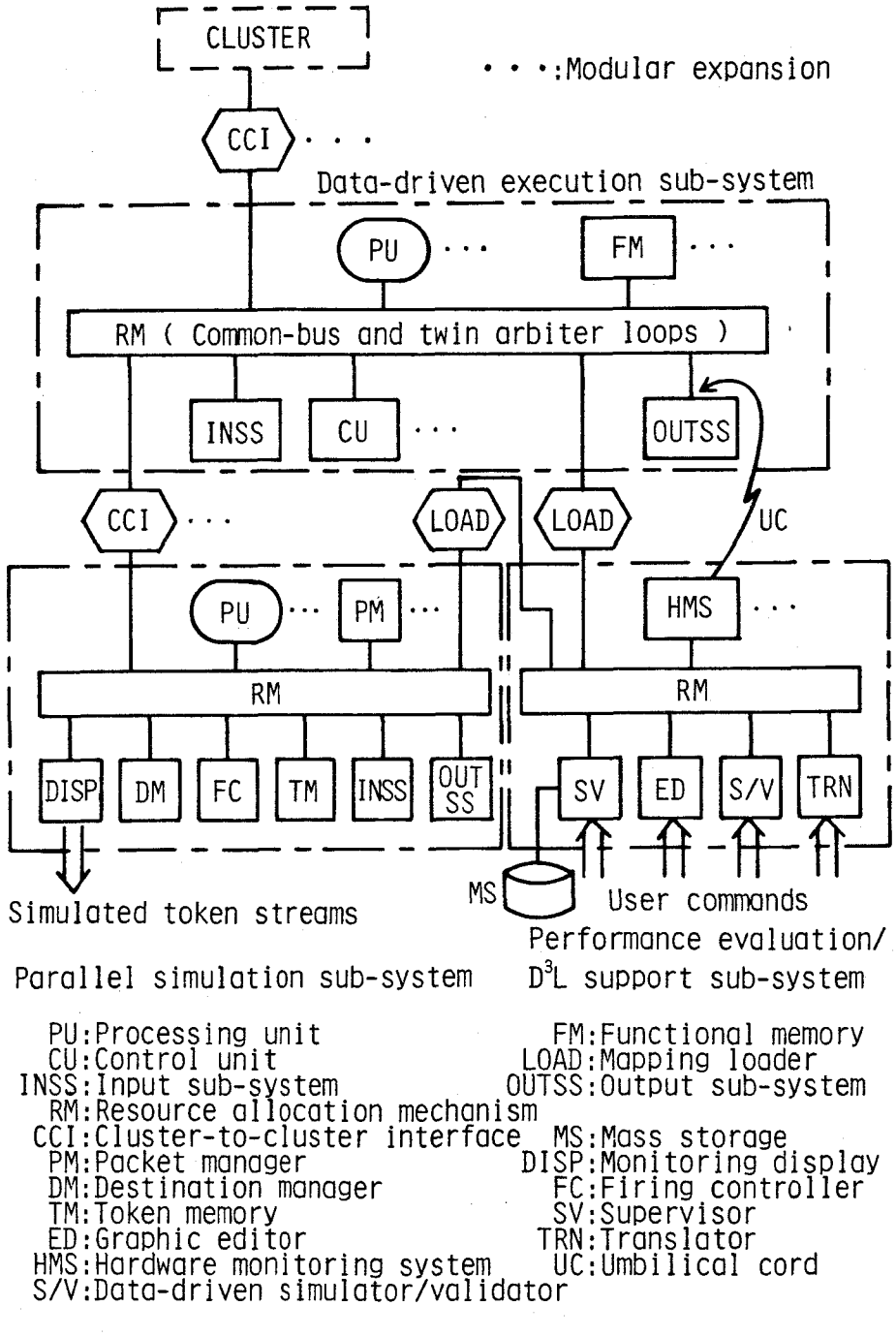


図5.3　実験システムの機能構成

Fig. 5.3－Functional organization in the experimental system.

いるが，実験システム自身の開発援助にも効力を発揮する。

⑥スーパバイザ（Supervisor ; SV）

プログラミング言語Cで記述された実験手続に従って，①～⑤のモジュール及び D^3L プログラムのデータ駆動形実行機能モジュールを制御する。

現在，本実験システムは D^3L プログラムの一実行形式，並びにこれらのサポートソフトウェアモジュールの実装によって，図5.3に示すように，

ⅰ）データ駆動形実行サブシステム，

ⅱ）並列シミュレーションサブシステム，

ⅲ）性能評価・ D^3L サポートサブシステム，

などからなる履歴依存性を含む高度並列処理方式の模索の強力なツールとして，D^3L の改良とその実験的検証に利用されている[29]。

5.3 実験結果とその検討

本節では，実験システム上の D^3L の一実行形式に関する基本的な実験結果について述べる。まず本実行制御方式におけるPU及びFM群における分散処理，CU群における分散制御の効果を確認する。次に，実験結果に基づき，対象とする D^3L プログラム構造を反映した階層的クラスタ構成をとる実験システム上に，本実行形式が基本的に高度並列処理を実現しうることを示す。

5.3.1 データ駆動形実行機能の負荷・機能分散

まず本方式におけるデータ駆動形処理実行のための基本機能の負荷・機能分散的実行の効果の確認と共に，本実験システムの μ_{cm}（⇨4.2.2）を求めるための基本的な実験を行った。対象として実験結果の検討の簡単のため，図5.4に示す手続を含まない斉一な関数的処理fからなるプログラム（Prog 1）を採用した。又，μ_{cm} を求めるため，関数の実行時間 T_f のばらつきが実験結果に影響を与えないことを確認した上で，T_f ＝100ms，10ms，1ms，0.1msの場合について実験を行った。更に，履歴依存性の影響の検討のため，図5.5に示す履

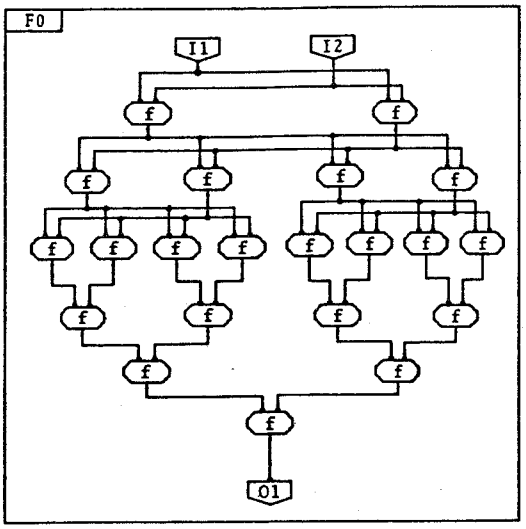


図5.4　D³Lによるモデルプログラム (1)

Fig.5.4－A model program in D³L (Prog 1).

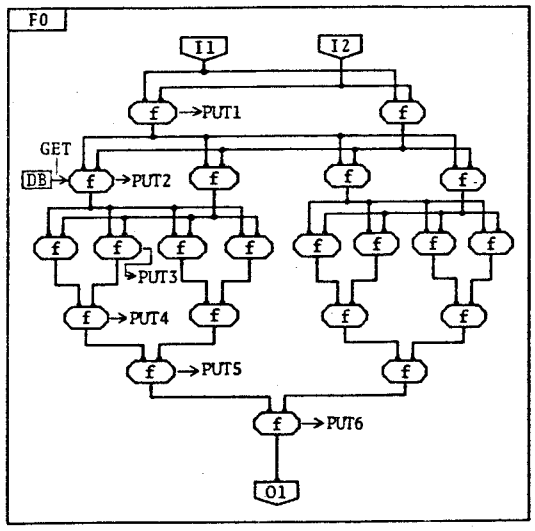


図5.5　D³Lによるモデルプログラム(2)

Fig. 5.5－A model program in D³L (Prog 2).

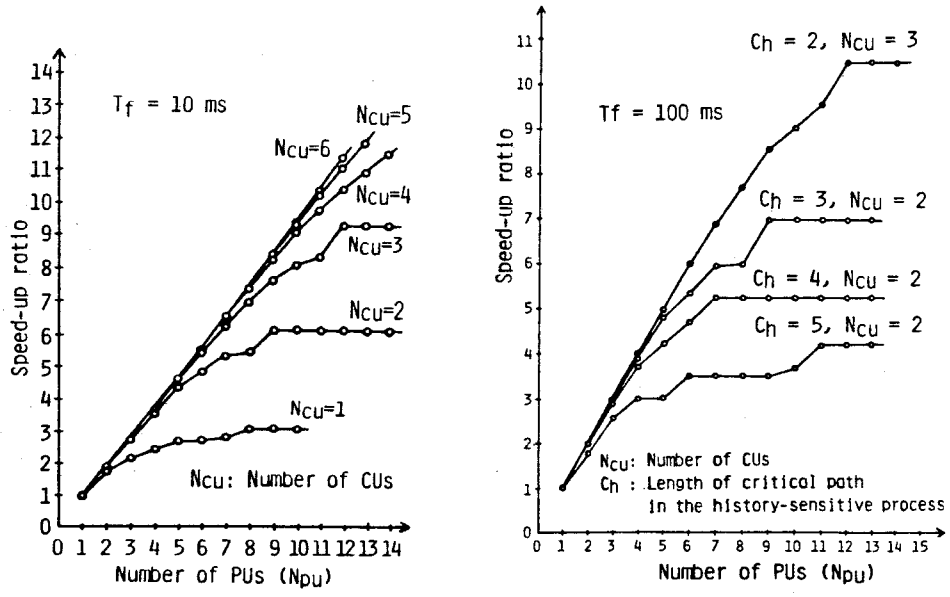
```
#define SBCTOTAL 17        /* Total number of SBCs */
#define CUMAX    10        /* Maximum number of CUs */
#define LOAD      1        /* ID code for Loader */
#define BUSM      2        /* ID code for Bus monitor */

#include  "mcp.h"          /* System vector table */

main ()
{
        int  cu, pu;
        mcpinit ("MCP.MCP");
        sendcom ( BUSM,"<busm.ini" );
        for ( cu = 1; cu <= CUMAX; cu++ ) {
                sendcom ( LOAD,"RESET 6" );
                sendcom ( LOAD,"%d S 0 CUDRV.COM @TEST.HEX", cu );
                for ( pu = 1; cu + pu <= SBCTOTAL; pu++ ) {
                        sendcom ( LOAD,"1 S 0 PUDRV.COM FUNC.HEX" );
                        sendcom ( BUSM,"R 2 10 /n" );
                }
        }
        mcpexit ();
}
```

図5.6　スーパバイザモジュールにおけるコマンドシーケンス例

Fig. 5.6－An example of command sequences in the supervisor module.

歴の参照（ＧＥＴ）及び更新（ＰＵＴ１～６）を含むプログラム（Prog 2）についても実験を行った。

Prog 1 に関して，ＰＵ・ＣＵ群における負荷分散の効果を検討するため，ＩＰを送信する入力ＳＳ，ＦＰを受信する出力ＳＳ，ＰＵ及びＣＵの機能を実験システム上の各プロセッサに専用的に割当て，ＣＵ数をパラメタとしてＰＵを順次増設し，出力されるＦＰ数の変化を，ＨＭＳを用いて測定した。この実験で用いたスーパバイザモジュールにおけるコマンドシーケンスを図5.6に示す。ＦＰ数によりシステムのストリーム処理能力Ｓが測定されるが，その評価基準として，オーバヘッドがない逐次処理システムの処理能力を１とした性能向上比（Speed-up ratio）を採用した。実験結果には，すべて同様の傾向がみられた。 T_f が10msの時の実験結果を図5.7(a)に示す。ＰＵ数 N_{pu} の増加に伴いそのＳの向上が飽和するが，その値はＣＵ数 N_{cu} に比例している。 従って，



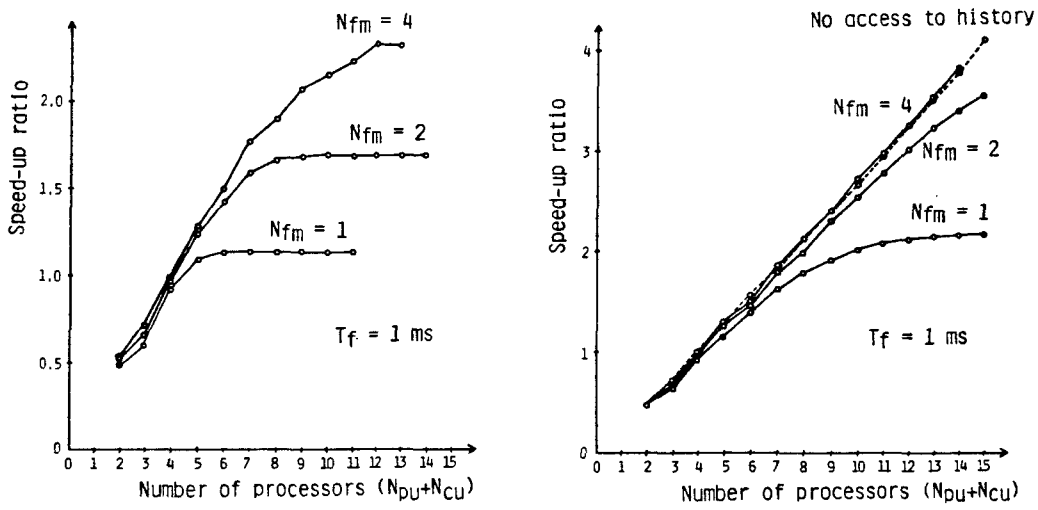
(a) History－insensitive program.　(b) History－sensitive program.

図5.7　ＰＵ群の負荷分散による性能向上

Fig. 5.7－Performance improvements by load－sharing in the PUs.

本方式におけるＰＵ群での分散処理およびＣＵ群での分散制御の効果が確認できる。

次に，Prog 2 に関して，図5.5のように履歴の参照（ＧＥＴ）を設定し，その更新がそれぞれＰＵＴ１～ＰＵＴ６の６つの場合に関して，実験システムの一つのプロセッサにＦＭの機能を割当てて同様の実験を行い，図5.7(b)に示すような結果を得た。即ち，ＰＵＴ１の場合は履歴の更新の後，その参照を行うため，世代間の相互作用いわゆる履歴依存性がないので，図5.7(a)と全く同様のＳの向上が得られたが，ＰＵＴ２～６の場合は，ＣＵの付与に伴うそのＳの向上がある一定の値で飽和する傾向がみられた。その値は，N_{cu} が１の場合の飽和したＳを１とするとＰＵＴ２～６の場合に対してそれぞれ，６，３，２，1.5，1.2であり，プログラム中のソース（I1，I2）からシンク（O1）に至るパイプラインの最大ステージ数，即ち，クリティカルパスのステージ数Ｃと履歴の参照および更新に挟まれるパイプラインのステージ数 C_h との比と一致する。又，飽和に至るまでのＳの向上は図5.7(a)と同様の結果を得た。更



(a) History－sensitive program.　　(b) History－insensitive program.

図5.8　ＦＭ群の機能分散による性能向上

Fig.5.8－Performance improvements by function－sharing in the FMs.

に，二つ以上の履歴を含むプログラムについても，同様の実験を行い，最大のC_hを持つ履歴が支配的に作用すること，及び図5.8(a)(b)に示すように複数のＦＭへの履歴の分割配置によるＳの向上を確認している。従って，プログラムが履歴依存性を含む場合においても，ＦＭ群へ各履歴を分割配置し，その参照・更新をクラスタ内に局所化すれば，ＰＵ，ＦＭ群での分散処理およびＣＵ群での分散制御の効果によって，本方式が基本的には高度並列処理を実現しうることがわかる。

5.3.2　システム規模とその性能向上

システム構成規模とそのＳの向上におけるT_f，機能構成比の影響を検討するために，Prog 1 に関する各実験結果からプロセッサ数一定の条件下で最大のＳを示したＰＵ・ＣＵ構成を求めた結果を図5.9に示す。図中の添字はそれ以降のシステム構成におけるN_{cu}を示す。各々のシステム構成におけるＳの向上比とN_{pu}よりＰＵ群の実効的な稼働率を求めると，T_f = 100ms, 10ms, 1ms, 0.1ms に対し，各々ほぼ100％，90％，50％，10％であり，本実験システムのＣＵ群とＲＭによるＰＵ上の関数駆動に要する時間は約1 ms と推定される。従って，（μ_{pm}／μ_{cm}）即ち関数駆動に要する制御オーバヘッドとT_fの比（Overhead ratio）をｒとすると，T_f = 100ms, 10ms, 1ms, 0.1ms は各々ｒ = 0.01, 0.1, 1, 10 となる。

又，Prog 1 に関して，ＣＵ機能を持つ各プロセッサにその副プログラムのデータ従属性を示す接続構造を二つ以上重複させて割当て，複数の世代の異なるデータの発火制御をパイプライニングさせた場合の実験結果を図5.10に示す。Ｓの向上比は，ｒ＝１の時はＣＵが過負荷になり当然低下するが，ｒ＝0.1の時，２世代のデータをパイプライニングさせた場合まで増加した。
Prog 1 の平均的に同時に並列実行可能な処理数，即ち，4.2.2に述べたＰは関数的処理数21，クリティカルパス６より，3.5となる。従って，３世代のデータのパイプライニングにより，そのＣＵに割当てられる副プログラムの規模が実効的に３倍になり，$P \cdot \mu_{pm} > \mu_{cm}$，即ち，適正なプログラム規模より

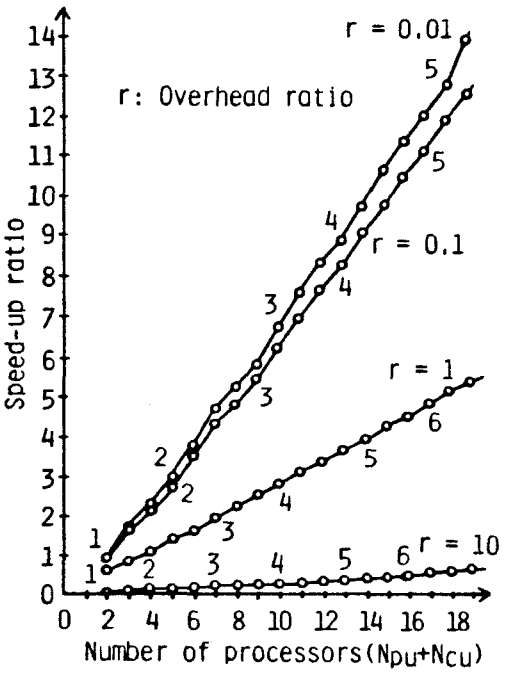


図5.9　モジュラなシステム拡張による性能向上

Fig. 5.9－Performance improvements by modular system expansion.

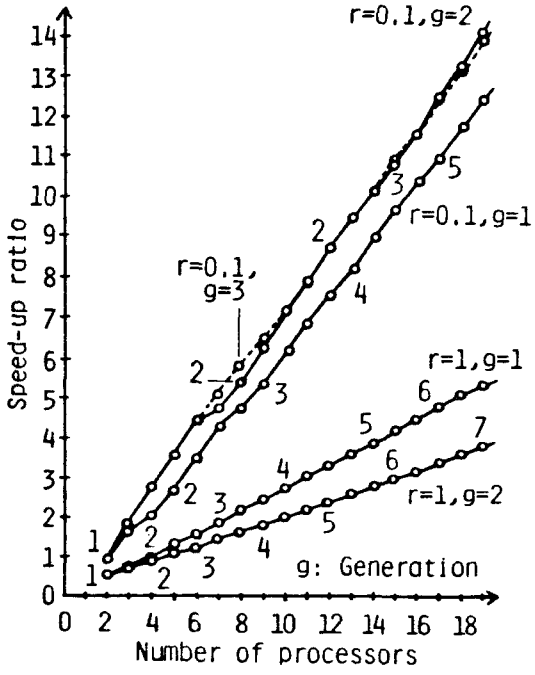
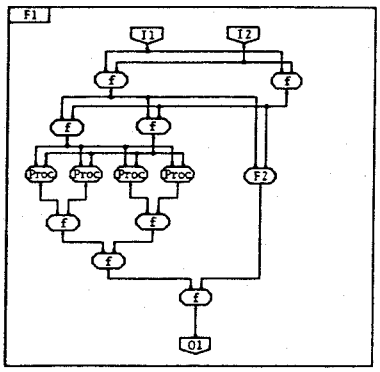


図5.10　パイプライニグされたＣＵ機能による性能向上

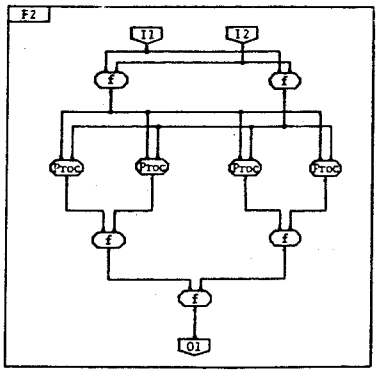
Fig. 5.10－Performance improvements by pipelined firing control function.

大きくなる。故に，この実験結果は，4.2.2に示した平衡条件を裏付けると共に，Prog 1は 1クラスタからなる本システムで実行可能なプログラム規模であることを示す。

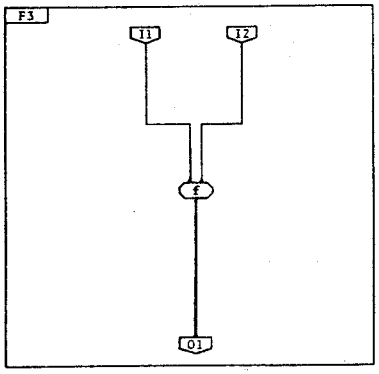
最後に，大規模なプログラムを副プログラムに分割し，階層的なクラスタ構成をとるシステムを用いて実行した場合の実験について述べる。対象として，次の3プログラムを用いた。即ち，図5.11(a)(b)中の手続 Proc に，1）手続Ｆ3を割当てた Prog 3（Prog 1と同等のプログラム規模），2）手続Ｆ4を割当てた Prog 4，3）手続Ｆ5を割当てた Prog 5。これらをトランスレータを用いて展開して，1クラスタ及び2クラスタからなるシステムにより実行さ



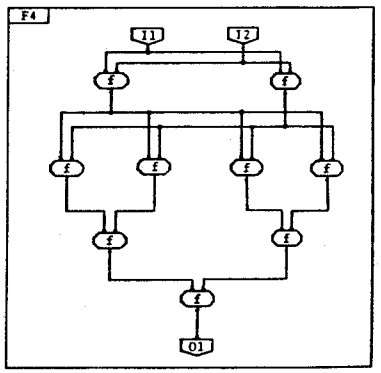
(a) Procedure F1.



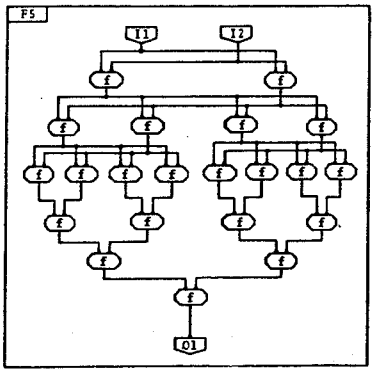
(b) Procedure F2.



(c) Procedure F3.



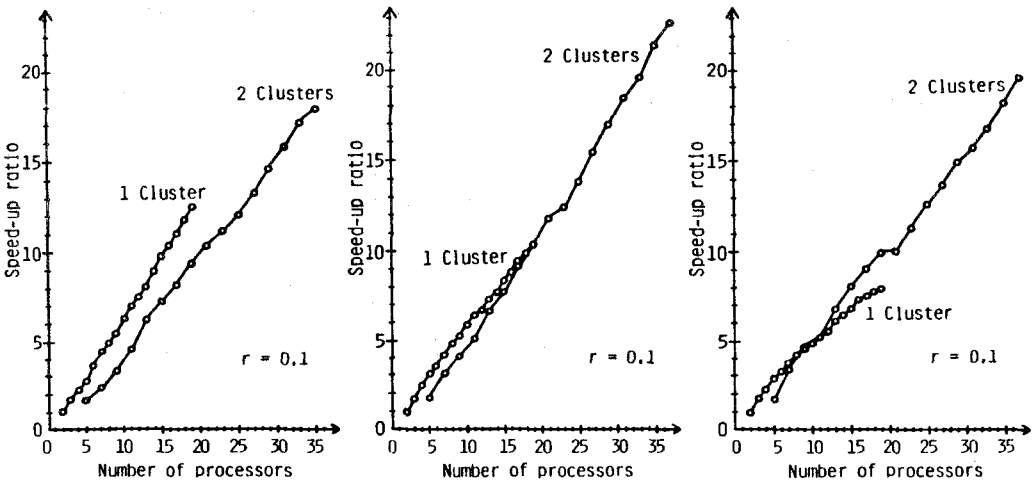
(d) Procedure F4.



(e) Procedure F5.

図5.11　D^3L によるモデルプログラム (3)

Fig.5.11－A model program in D^3L (3).



(a) Prog 3.　　(b) Prog 4.　　(c) Prog 5.

図 5.12 階層システムにおける性能向上

Fig. 5.12－Performance improvements in the hierarchical system.

せた。階層系では，上位，下位クラスタに図中に示す手続，F 1，F 2 を副プログラムとして，それぞれ割当てている。実験結果を図 5.12 (a)～(c)に示す。

即ち，Prog 3 では，先の実験結果よりわかるように，1 クラスタで実行できるプログラム規模であるため，階層系による分割損が生じ，S の向上比が 1 クラスタによる実行結果より低下した。又，Prog 5 では，分割による S の向上がみられるが，上位，下位クラスタに割当てられるプログラム規模が関数的処理数（92，89），クリティカルパスのステージ数（11，9）と大きいため，各クラスタに割当てられた副プログラムでは，$P \cdot \mu_{pm} \geqq \mu_{cm}$ となり，CU上でのデータの滞留が生じ，各々のシステム構成における S の向上比と N_{pu} より求めた実効的な PU 群の稼働率は約 70 %を示した。Prog 4 では適正な副プログラム分割，即ち，上位，下位クラスタに割当てられる関数的処理数（44，41），クリティカルパスのステージ数（9，7）となり，実効的な PU 群の稼働率は，図 5.9 の Prog 1 に関する 1 クラスタの実験結果（r＝0.1）と同程度

の約90％を示した。

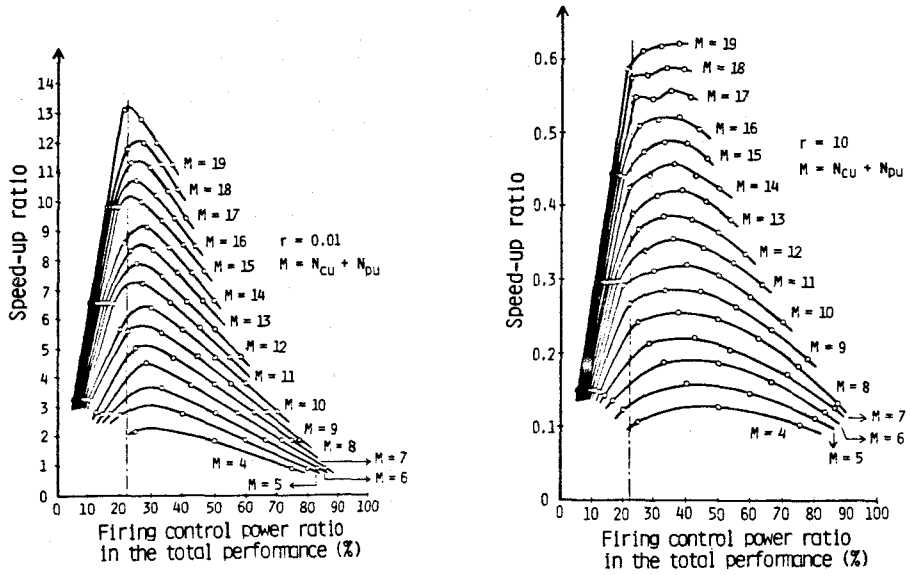
Prog 1～5 に関する実験結果から，適正な T_f の設定，プログラムの分割配置によって，$P \cdot \mu_{pm} \leqq \mu_{cm}$ を保証すれば，本方式の機能・負荷分散が，対象とする D^3L プログラムの並列性が許す限り，システム資源に比例したSの向上，即ち高度並列処理を実現できることがわかる[31]。

5.4 D^3L による高度並列処理実現手法

本節では，まず D^3L による高度並列処理の実現のための条件について考察し，D^3L プログラムが陽に示す並列処理構造を余すことなく活用できるデータ流が確保されるためには，データの追越しが生じない実行環境が必要であることを明らかにする。次に，本方式では4.2.2にふれた D^3L プログラム並列処理構造P及び5.3.2に定義したオーバヘッド比rから，そのような実行環境を提供する適正な基本機能構成比が求めうることを示すと共に，対象とする D^3L プログラム構造に基づく機能割当手続を提案している。

5.4.1 高度並列処理の実現に対する考察

既に，データ駆動原理による高度並列処理の実現には，D^3L プログラムが陽に示すアルゴリズムに基づく並列処理だけでなく，実行時に動的負荷分散を行い，システム内に滞留のないデータ流の確保が必要なこと（⇨4.2.1）を示し，システムの動作モデルによる解析から，前提条件として，ⅰ）$\mu_{pm} << \mu_{cm}$ を考慮した並列実行されるタスクレベルの設定，ⅱ）$P \leqq (\mu_{cm} / \mu_{pm})$ を保証する D^3L プログラムの分割割当（⇨4.2.2），を明らかにした。本実行制御では，D^3L の言語仕様に従った基本機能分割，データ駆動原理を自然に実現できる資源割当モードを採用しているので，高度並列処理を実現しうる状況では，対象とする D^3L プログラム構造に従った理想的なデータ流が確保されていると予想される。この時，システムの機能構成は対象とする D^3L プログラムの構造に適した一定の比になると考えられる。図5.13にProg 1に関する実験結果より求めた本方式における処理・制御機能比と性能向上比の関係を示



(a) $\mu_{pm} << \mu_{cm}$.　　(b) $\mu_{pm} >> \mu_{cm}$.

図 5.13 基本機能構成比と性能向上

Fig. 5.13－Basic functional organization ratio and performance improvements.

す。図 5.13(b) のようにシステムが（$\mu_{pm} >> \mu_{cm}$）状態にあり，高度並列処理を実現しえない場合は性能向上における処理・制御機能比の影響が顕著に現れないが，図 5.13(a)に示す高度並列処理方式を実現しうる（$\mu_{pm} << \mu_{cm}$）時，システムの性能向上のピークはある一定の処理・制御機能比においてみられる。これは，オーバヘッド比 r が充分小さく並列実行の制御に要する時間が無視できる場合と考えられ，(i)及び(ii)の二つの前提条件を満足し，処理資源の投入に見合う性能向上が実現されることは容易に想像できる。

しかし，実際のシステムでは，一般的には制御オーバヘッドは充分小さくできるが無視できない。このような状況において，制御オーバヘッドと並列実行されるタスクレベルとにどのような条件が必要となるかについて考察する。

実行制御のオーバヘッドが無視できるような理想的なシステムでは，実行可能となった多数の処理はすべて同時に並列実行される。しかるに一方，そのオーバヘッドが充分小さいが無視できない現実のシステムでは，同時に実行可能

な処理が多数発生した時，制御能力を越える処理の発火制御には時間遅れが生じる。従って，この時システム内にデータが一時的に滞留する。D^3L プログラムが陽に示すデータ従属性の上のデータ流をイメージすると容易に理解されるように，このようなデータの滞留が生じないためには，実行可能となる処理がそのデータ従属性に基づく半順序で生起し，順次余すことなく駆動されることが必要である。即ち，図3.9に示したようにデータの追越しがない実行環境でなければならない。特に，D^3L プログラムは履歴依存性を許しているため，その副作用の回避のための参照・更新規則に従ったアクセスを行うためには，データの追越しが生じる実行環境では，待ち合わせが起きることは想像するに難くない。図5.7(b)に示した実験結果は履歴へのアクセスにおけるデータの滞留が生じない状況までシステムの性能向上を実現しうることを示している。このような実行環境を実現するためには，制御のオーバヘッドを T_c，並列実行される各処理に要する時間を T_p，理想的なシステムにおいて同時に実行可能となる処理数を N_i とすると，これらの間に次の関係が満たされなければならない。

$$N_i \cdot T_c < T_c + T_p$$

即ち，最初に発火する処理は T_c 後に実行され，$(T_c + T_p)$ 後にその実行を終了するから，この時間内で同時に実行可能なすべての処理の発火制御が行われる必要がある。これを変形すると，

$(N_i - 1) \cdot T_c < T_p$ となる。

図5.10 に示した実験結果は，この条件を裏付けていると考えてよい。又，実験では結果の検討の簡単のため，PU群上の関数的処理の実行時間を斉一としたので，上述の条件で充分であるが，データの追越しが生じない実行環境には，言うまでもなくそのばらつきTは次式の範囲でなければならない。

$$T < N_i \cdot T_c$$

5.4.2 D^3L プログラム構造に基づく機能構成法

本方式は，その処理および制御機能の構成比が適正であれば，システム構成

規模にほぼ比例した性能向上，即ち高度並列処理を実現できる。しかし，高度並列処理方式の適用分野となる大規模なシステム構成を想定すると，その機能構成比を実験的に求めることは困難となることが容易に予想される。従って，その機能構成を求めるなんらかの指針が必要となる。

本実行制御方式では，制御ネック（$N_{cu} << N_{pu}$）の時N_{cu}に，処理ネック（$N_{cu} >> N_{pu}$）の時N_{pu}に比例して，各々の状態におけるストリーム処理能力（S_c，S_p）が向上し，処理・制御機能の構成比が適正であるとき，$S_c = S_p$となると考えられる。ここで，本システムにおける各CUのPU群上の各処理の駆動に要する時間をT_cとすると，S_c，S_pは次のように求められる。

①$N_{cu} << N_{pu}$，即ち，制御ネック状態の場合，

ランクiに属する関数的処理数をN_iとする。但し，ランクを次のように定義する。ソース（Source）のランクは0とし，先行するノードのランクのうち最大のものがiのノードのランクをi＋1とする。即ち，シンク（Sink）を除くノードのランクのうち，最大のものがCと一致する。この時，iに属するすべての処理の実行の完了には，時間$T_i = N_i \cdot T_c + T_f$を要する。従って，システムの応答時間$T_r$は，

$T_r = \sum_{i=1}^{c} T_i = \sum_{i=1}^{c} (N_i \cdot T_c + T_f) = N \cdot T_c + C \cdot T_f$，故に，

$S_c = N_{cu} / T_r = N_{cu} / (N \cdot T_c + C \cdot T_f)$で与えられる。

②$N_{cu} >> N_{pu}$，即ち，処理ネック状態の場合，

この時は，T_fが$(T_f + T_c)$になった場合と等価であるので，

$T_r = \{ N_{cu} \cdot N \cdot (T_f + T_c) \} / N_{pu}$となり，

$S_p = N_{cu} / T_r = N_{pu} / (N \cdot T_f + N \cdot T_c)$で与えられる。

従って，適正な処理・制御機能の構成比は$S_c = S_p$より

$(N_{cu} / N_{pu}) = (N \cdot T_c + C \cdot T_f) / (N \cdot T_f + N \cdot T_c)$で求められる。

ここで，Tfは関数的処理の実行時間，N_{pu}，N_{cu}はPU数，CU数，N，CはD^3Lプログラム中の関数的処理数，クリティカルパスのステージ数である。

更に，$T_c / T_f = r$，$P = N / C$を用いると，

表5.1　データ駆動形システムの最適基本機能構成比

(a) Prog 1.

M	実験値	理論値
3	1, 2	1, 2
4	1, 3	1, 3
5	2, 3	1, 4
6	2, 4	2, 4
7	2, 5	2, 5
8	2, 6	2, 6
9	3, 6	2, 7
10	3, 7	3, 7
11	3, 8	3, 8
12	3, 9	3, 9
13	4, 9	3, 10
14	4, 10	4, 10
15	4, 11	4, 11
16	4, 12	4, 12
17	5, 12	4, 13
18	5, 13	5, 13
19	5, 14	5, 14

(b) Prog 4.

M	実験値	理論値
5	1, 1 : 1, 1	1, 1 : 1, 1
7	1, 2 : 1, 2	1, 2 : 1, 2
9	1, 3 : 1, 3	1, 3 : 1, 3
11	2, 3 : 2, 3	1, 4 : 1, 4
13	2, 4 : 2, 4	1, 5 : 1, 5
15	2, 5 : 2, 5	2, 6 : 1, 5
17	2, 6 : 2, 6	2, 7 : 1, 6
19	3, 7 : 2, 6	2, 7 : 2, 7
21	3, 7 : 2, 8	2, 8 : 2, 8
23	3, 8 : 3, 8	3, 9 : 2, 8
25	4, 8 : 3, 9	3, 10 : 2, 9
27	4, 9 : 3, 10	3, 11 : 2, 10
29	4, 10 : 3, 11	3, 11 : 3, 11
31	4, 11 : 3, 12	3, 12 : 3, 12
33	4, 12 : 3, 13	4, 13 : 3, 12
35	4, 13 : 3, 14	4, 14 : 3, 13
37	4, 14 : 3, 15	4, 15 : 3, 14

$(N_{cu}/N_{pu})=(N\cdot r+C)/(N\cdot r+N)=(P\cdot r+1)/(P\cdot r+P)$ ……(1)

となり，適正な構成比は2.2に述べたように，D^3L プログラム の並列処理構造Pと処理レベルと制御オーバヘッドの比rで決定される。ここで，Prog 1 ($r=0.1$，$N=21$，$C=6$)に関して，図5.9の実験結果における機能構成比と(1)式を適用して求めた機能構成比の比較を表5.1(a)に示す。表中のMはそのシステム構成における資源数($N_{cu}+N_{pu}$)を示す。表5.1(a)では，構成比をN_{cu}，N_{pu}の形式で与えているが，各モジュール数は±1の精度で一致している。

又，階層系については，プログラム全体および副プログラムのクリティカルパスのステージ数をC，C_s，上位，下位クラスタに割当てられる D^3L 副プログラム中の関数的処理数をN_m，N_s，CU数をN_{cum}，N_{cus}とすると各クラスタ中のデータ流は，各々，$N_{cum}/(N_m\cdot r+C)$，$N_{cus}/N_s\cdot r+C_s)$ となり，システム全体のデータ流が保存することから，

$N_{cum}/(N_m\cdot r+C)=N_{cus}/(N_s\cdot r+C_s)$ 故に，

$N_{cum}/N_{cus}=(N_m\cdot r+C)/N_s\cdot r+C_s)$ ……(2)となる。

ここで，二つのクラスタからなる階層系を想定したが，より大規模な三階層以上のクラスタ構成なるシステムについても，ＣＣＩによって接続される二つクラスタからなる階層に着目し，(2)式を順次適用することができる。従って，任意のクラスタ構成における適正な機能構成比は，対象とする D^3L プログラムの幾何学的接続構造に基づき，次の手続により求められる[31]。

1° (2)式より，各クラスタのＣＵ数の比を決定する。

2° (1)式より，各クラスタ中のＣＵ数とＰＵ数の比を決定する。

即ち，二つのクラスタから構成される階層系では，次の４式を解くことにより適正な機能構成が得られる。

$M = N_{pum} + N_{cum} + N_{pus} + N_{cus} + N_{cci}$,

$N_{cum} / N_{cus} = (N_m \cdot r + C) / (N_s \cdot r + C_s)$,

$(N_{cum} / N_{pum}) = (N_m \cdot r + C) / (N_m \cdot r + N_m)$ 及び

$(N_{cus} / N_{pus}) = (N_s \cdot r + C_s) / (N_s \cdot r + Ns)$。

但し，N_{cci} はシステム内のＣＣＩ数で，この場合は１となる。又，Npum，N_{pus} は上位，下位クラスタ内のＰＵ数である。ここで，Prog 4（$r = 0.1$，$C = 9$，$N_m = 44$，$C_s = 7$，$N_s = 41$）に関して，図 5.12(b) に示した実験結果における機能構成比とこの手続により求めた機能構成比の比較を表 5.1 (b) に示す。この表では，構成比を N_{cum}，N_{pum}：N_{cus}，N_{pus} の形式で与えているが，各モジュール数は±１の精度で一致している。

この手続を求める過程において，履歴依存性のない場合を想定したが，履歴依存処理を含む場合にも，各履歴の参照・更新をクラスタ内に局所化させた上で，図 5.7 (b) に示したProg 2に関する実験結果により各クラスタ中のＣＵ数の上限を決定し，この手続を適用できる。又，T_f を斉一として扱ったが，そのばらつきが 5.4.1 に述べた範囲に含まれる場合はこの手続が有効であること及び，そのばらつきが大きい場合には，その関数をパイプライニング方向に分割した新たな並列処理構造に基づきこの手続を適用できることも実験的に確認している。

5.5 結　言

本章では，第4章に述べた D^3L によるデータ駆動原理の実現法に基づき，共通バス形式のマルチプロセッサ構成をとる実験システム上の，一実行形式ならびにそれをサポートする D^3L の処理系を述べると共に，本方式が，対象とするプログラム構造・規模に適したクラスタ構成および適正な処理レベルの設定により，高度並列処理を実現しうることを示した。更に，対象とする図的プログラムの幾何学的接続構造に基づき，本方式の階層的なクラスタ構成上に各基本機能を分割配置するための指針を半定量的に提案した。

本章では，PU群における機能分散，即ち，D^3L プログラム中の各関数的処理の生起頻度を考慮した機能割当の効果についてはふれなかったが，GPSSによるシミュレーション[67]により良好な結果を得ると共に，現在，本システムを用いて更に詳細な実験を続けており，これらについても，一応の見通しを得ている。

本章に述べた実験システム上の実行形式では，D^3L プログラムに陽に示される並列処理構造に基づき適正な基本機能の構成比が近似的に求められると共に，各機能のモジュラな増設・削除によってその機能構成を実験的に検討できる。従って，本方式とその実験システムは，ある特定の問題向きの専用的システムを構成する際の設計指針を提供することが期待される。

又，実験結果は，問題に内在する履歴依存性がデータ駆動原理に基づくシステムの処理能力向上に限界を与えることを示している。現在，このような履歴依存性をより効果的に実行するため，データ駆動原理に時刻の概念を導入したイベント駆動方式の導入を検討している。

第6章　結　論

本論文では，高度並列処理方式の模索のために，言語主導形の実現手法を採用し，高度並列処理構造に関して最適な表現形式を採用した中間言語の一つとして，D^3Lを提案すると共に，これにより，1）高度並列処理に必要な同時並行／パイプライン処理を含む並列処理構造が陽に記述できること（了解性），2）履歴依存性を有する処理に関しても副作用のない実行が保証されること（検証性），3）D^3L の仕様に基づく自然な実行形式を採用すれば，与えられた問題の構造・規模に適応できる拡張性を持つ実行制御方式が実現可能なこと（階層性），並びにその実験的検討によって，4）この D^3L の一実現法が高度並列処理を実現しうること，5）高度並列処理方式における基本機能構成が対象とする D^3L プログラムの幾何学的接続構造より近似的に求まること，などについて考察した。

まず本研究で得られた成果および今後に残された問題を簡単にまとめておく。

第2章においては，高度並列処理方式の実現のために，従来のハードウェア先行形のシステム開発過程の反省に基づく，次のような言語主導形の実現手法を述べた。即ち本研究では，高度並列処理に関して最適な表現形式を採用した言語を，問題向き利用者言語と実ハードウェアの間に位置する中間言語としてまず定式化し，これに副作用を回避できる厳密な実行規則を付与すると共に，最終的にこの言語の実行規則を最も効率よく実現する並列処理機構を模索する方針を採用した背景を論じた。まず高度並列処理方式の満たすべき要件として，1）対象とする問題に内在する並列性を余すことなく活用できる柔軟な機能および負荷分散処理方式，2）分割損のない効果的な分散制御原理，並びに3）システムと外界との効率の良い入出力インタフェース，を明確にした。次に，高度並列処理方式の実現において，従来の文章形記述に代わる図的表記に基づく中間言語およびそれにより表現される履歴依存性を許すデータ駆動原理が有効であることを示した。又，これらが現状のソフトウェア危機への対処，利用者言語あるいは仕様記述過程の定式化に関しても有望であることに言及すると

共に，そのための検討事項として，図式中に陰に含まれる履歴依存性の検出とそれを中間言語としての D^3L の仕様に変換する手法，並びに D^3L への抽象データ構造の導入にふれた。

第3章においては，副作用の生じない並列処理構造を陽に記述できる D^3L を提案した。まず D^3L に採用した図的表記法，即ちデータ駆動図式の仕様を示すと共に，D^3L によって記述される階層的なプログラムに副作用が生じない保証を与える検証性について述べた。次に，履歴依存処理の記述のため拡張されたデータ駆動図式についても同様の検証性が保証されることを示すと共に，図式中の履歴の具体的実現例として，最小限の履歴のコピーによって，履歴依存性を許すデータ駆動形実行環境における副作用を回避できる参照・更新規則を持つタグ付き記憶セルを提案した。

第4章においては，D^3L によって表現されるデータ駆動原理の一実現法について述べた。まずデータ駆動原理による高度並列処理の実現には，D^3L プログラムが陽に示すアルゴリズムに基づく並列処理のみならず，入力ストリームに応じたデータ流をシステム内に確保できる実行時の動的な負荷分散が必要であることにふれた。次に，システムの動作をモデルにより解析し，高度並列処理実現のための前提条件として，1）対象とする D^3L プログラムの分割割当，及び2）並列実行されるタスクレベルの設定，の適正化を示した。更に，D^3L プログラムのデータ駆動形実行のための基本機能群を示した後，これらの条件を満たす実行制御の一方式として，D^3L プログラムのブロック構造を反映した階層的なクラスタの繰返し構成をとるシステム上で，これらの基本機能群の負荷ならびに機能分散的実行を統一的に実現する実行制御手法を述べた。又，残された問題として，効果的な発火制御機構および対象とする問題の構造に応じた基本機能の分割・併合割当などがあることにふれた。

第5章においては，D^3L による高度並列処理の実現手法を実験的に検討した。まず共通バス形式のマルチプロセッサ構成をとる実験システムの特徴およびその上での D^3L の一実行形式ならびに処理系について簡単にふれた。次に，階層的システムへのプログラムの分割配置および並列実行されるタスクレベル

の設定が適正であれば，本方式が高度並列処理を実現しうることを実験結果に基づき示した。更に，対象とする D^3L プログラム構造に基づき，そのデータ駆動形実行のための各基本機能を階層的なクラスタ構成上へ分割配置するための指針を半定量的に示し，ある特定の問題向きの専用システムを構成する際の設計指針を与えた。

本研究の最大の成果は，D^3L を核とする言語主導形の方針が高度並列処理方式の実現手法として有望であると結論されたことにある。即ち，D^3L が中間言語の機能を果たしうると言うこの結論に従えば，今後の研究の方向としては，1）D^3L の理論的基礎の強化と実用性の改良，はもとより，2）問題向き利用者言語の検討，3）物理的並列処理機構の模索，が挙げられる。

D^3L はまず第一に，言語の理論的基礎として，履歴依存処理を含めて陽に高度並列処理構造を記述すると同時に副作用の回避も完全に保証することを意図した。更に第二に，表現上の記法として，人間の思考手段あるいは人間－機械インタフェースに馴染み易い図的表現による記述形式を採用した。

計算機言語は，自然言語と同様に，使用経験によって次第に洗練され表現力を豊かにする。FORTRANのように早期に確立した言語においても，今日なお新しい版が提供されつつあるのはその証拠である。D^3L の試みはいずれも新しいものであるだけに，理論的にも実用的にも多くの実験的検証を必要としている。D^3L がまず当面する最大の問題はプログラミングの基本的慣習の変革にある。言うまでもなく，言語は思考の道具である。計算機言語に関しても事情は同様であり，逐次処理言語に慣れ親しめば，逐次処理形思考からの脱出が非常に困難になり，遂には逐次的アルゴリズムの世界に埋没していることにさえ気付かない状態に陥り易い。しかし，一旦並列処理の自然さに気付けば，思考そのものが並列的，多角的になり，並列アルゴリズムを自在に活用しうる能力が回復されると期待される。並列思考は，直列思考を包含する，より自然な上位の思考形態だからである。しかしながら，思考の変革は必ずしも容易ではない。しばしば指摘されるように，逐次処理言語間の移行においても，単にソフトウェア資産蓄積による慣性効果だけでは説明しきれない，いわゆる初習言

語症候群（First language syndrome）が見られる。初めて習得した言語は，習熟によって自在になればなるほど，離れ難い執着心を生むものである。まして基本的な思考の転換を迫られれば，抵抗の姿勢はむしろ自然な防御反応であると言える。もし並列処理記述の道具が，抵抗感を増幅する，煩雑なものであれば，移行は殆んど成功しないであろう。D^3L のような中間言語でもその記述法ないし規則は親しみ易く自然でなければならない。従って，心理的に受入れ易い記述形式を開拓することが本研究の大きな課題として残されている。

更に，D^3L を基礎に問題向き利用者言語あるいは仕様記述過程を定式化するためには以下のような検討が必要となる。即ち，プログラミングの過程はもともと曖昧に形成された着想を，次第に曖昧さを排除して，厳密化する作業であると考えられる。即ち一般にはトップダウンの方向に概念を精密化し，より下位の層の検討によって上位の層の仕様を更に精密化する過程を反復して，全体の仕様が，最終的に決定されると考えてよい。例えば，極めて大まかな直感的ブロック図から出発し，更にこれをやや詳細な中ブロック図に分解して検討すれば，上位ブロック図で曖昧であった仕様に対するより厳密な再定義が求められる。この過程の繰返しによって設計が進行すれば，最終的には最下位層に到達する。有用な計算機言語は，このような過程を援助する機能を持たねばならない。本研究では，従来いわゆるＣＡＤの領域に属していた概念を含めて，一貫した言語体系として問題解決過程を捉えうる利用者言語の完成を重要な目標としている。計算機は与えられたアルゴリズムを実現する機械であるから，与えられた問題を計算機に実行させるにはまず，アルゴリズムの発見が必要となる。しかしアルゴリズムの発見は，特に計算機に作業させる場合に限らず，人間にとって永遠の難問である。現在の計算機ソフトウェアの基本的問題点の一つは，かりにアルゴリズムが発見されても，それを機械が理解する言語への翻訳過程に多くの問題が残されていること，即ち計算機言語と人間の思考とのギャップに深刻な問題がある。人間の自然な思考は一般的には並列あるいは図的であり且つ曖昧に始まりしだいに精密化される。しかしながら，現在の計算機言語は全体を直列に，且つ始めから精密に記述することを求める。この背反

を埋めるべく多くの仕様記述言語が提案[68)-(70)]されたにもかかわらず依然として広く利用されないのは，これら言語が直列形の形式的文章記述，即ち当初からの精密な記述を求めたことに起因している。本来，仕様記述は結果であって出発点ではない。本研究では，図的表現の階層的表現能力を利用して，このギャップを埋める設計支援システムの完成をめざしている。

又，物理的並列処理機構の形成には，言語の実行規則の実現，即ち実行制御方式の確立が最も重要な条件となる。現存の逐次処理形計算機に見られるように，同一の範疇に属する言語を用いても，その実行制御方式の実現法の相違から多くの機種が生まれる。データ駆動言語を実行する機械にも同じ事情があり，物理的高度並列処理機構の実現には，実行制御方式の詳細な検討が不可欠である。

D^3L プログラムの実行制御方式の高効率化は，既にふれたように結局発火可能な処理を高速に発見する手段の工夫，即ち，対となるデータの組をなるべく簡単な機構で高速に発見する問題に帰着する。無限の資源のもとで D^3L プログラムを実行する場合にはこの機構だけで充分であるが，資源の制約のある物理的高度並列処理機構上では必ずしも充分ではない。例えば資源の制約の下でデータの流れを最大にするためには，簡単なプログラムでも，適当なプログラム 分散配置による最適化あるいは実行順序の制御を必要とすることが予想される。特に履歴依存処理を有限の資源つまり記憶量の制約の下で実行する場合には，デッドロックが生じる可能性がありうる。従って一般に，与えられた問題を限られた資源で最も効率よく実行させるために，適当な実行制御順序の決定，即ち初期機能割当およびその資源割当動作モードの選択が極めて重要となることが多いと予想できる。従って，本論文に述べたデータ駆動原理の一実現手法，並びにその実験結果に基づく機能割当手続は，そのモデルの精密化をはかることにより，今後の研究の有効な指針となると考えられる。

本研究は上述のように，D^3L のような図的言語を導入して新しい言語体系を創造し，更に最終的にはこの言語を実行する物理的並列処理機構の模索を目標としている。これらの目標を実現するためには，実験的検討を要する課題が多く残されていることは言うまでもない。しかしながら，ＶＬＳＩ技術を背景

に，計算機システムを取り巻く技術は大きな変革を余儀なくされている。例えば，対象とする問題の要請とデバイスの技術的制限によって決定される計算機アーキテクチャにしても，従来は後者，即ち半導体デバイスの技術的制限に支配されてきたが，ＶＬＳＩの登場によって今後はプログラミング言語を含めたその設計ツールの能力に依存することが次第に明らかになってきている。既に述べたように，計算機の本質的な特徴はプログラム可能であることにある。従って，本論文に述べた，プログラミングの容易性を主眼とした図的言語を核とする高度並列処理方式の実現手法は，これらの要件を満たすものとしても，今後一層の発展が期待される。

謝　　辞

本研究の全過程を通じて，終始懇切な御指導，御鞭撻を賜った大阪大学工学部寺田浩詔教授に衷心より感謝の意を表する。

本論文作成にあたり，懇篤なる御指導いただくと共に種々の御高配を賜った大阪大学工学部手塚慶一教授，ならびに児玉慎三教授に深謝の意を表する。

大学院前期，後期両過程において電子工学一般および各専門分野に関し御指導，御教示を賜った電子工学教室中井順吉教授，小山次郎教授，尾崎弘名誉教授，電子ビーム研究施設裏克己教授，塙輝雄教授，産業科学研究所中村勝吾教授，角所収教授，ならびに松尾幸人名誉教授に深謝する。

本研究に関して，適切な御助言をいただいた大村皓一助教授，ならびに常に温かい御助言，御協力をいただいた浅田勝彦博士に深く感謝する。

本研究を進めるにあたり，常に温かい御激励をくださった大阪電気通信大学学長菅田栄治大阪大学名誉教授，筑波大学甘田早苗教授，長崎総合科学大学橘啓八郎教授，ならびに本学産業科学研究所黒田司助教授に厚く感謝する。

本研究に関し，有益な御討論および御助言をいただいた米国タータン研究所 William A. Wulf 博士，マサチューセッツ工科大学 Arvind 教授，ＩＢＭワトソン研究所 Tilak Agerwala 博士，カリフォルニア大学デービス校 James R. McGraw 教授，同大アーバイン校 Lubomir Bic 教授，ならびに英国東アングリア大学 M. R. Sleep 教授に心から謝意を表する。

筆者の属している寺田研究室の笹尾勤博士，江木康雄技官，大学院学生，西村仁志氏，岡崎信一郎氏，山内秀樹氏，金沢靖之氏，芳田真一氏，本学卒業生，日本電信電話公社石川啓二博士，関西電力株式会社磯修氏，新日本製鉄株式会社国本衛氏，横河北辰電機株式会社坂口和彦氏，帝人株式会社矢野哲也氏，また同研究室の岩井香保里嬢には種々の面で御協力いただいた。ここに記して感謝する次第である。

参　考　文　献

(1) Despain, A. M. and Patterson, D. A. : " X-TREE ", Proc. 5th Int'l Symp. Computer Architecture, pp. 141-151 (Apr. 1978).

(2) Patterson, D. A., et al. : " Design considerations for the VLSI processor of X-TREE ", Proc. 6th Int'l Symp. Computer Architecture, pp. 90-100 (Apr. 1979).

(3) Mead, C. A. and Conway, L. : " Introduction to VLSI System ", Addison-Wesley Publishing Co. (1980).

(4) Foster, M. J. and Kung, H. T. : " The design of special-purpose VLSI chips ", Computer, 13, 1, pp. 26-40 (Jan. 1980).

(5) Budzinski, Linn and Thatte : " A restructurable integrated circuit for implementing programmable digital systems ", Computer, 15, 3, pp. 43-54 (Mar. 1982).

(6) Patterson, D. A. and Sequin, C. H. : " A VLSI RISC ", Computer, 15, 9, pp. 8-21 (Sep. 1982).

(7) Treleaven, P. C. and Hopkins, R. P. : " A recursive computer architecture for VLSI ", Proc. 9th Int'l Symp. Computer Architecture, pp. 229-238 (Apr. 1982).

(8) Treleaven, P. C. : " VLSI processor architecture ", Computer, 15, 6, pp. 33-45 (June 1982).

(9) Tesler, L. G. and Enea, H. J. : " A language design for concurrent process ", Proc. SJCC, 32, pp. 403-408

(10) Comte, D., Durrieu, G. and Gelly, O. : " Parallelism, control and synchronization expression in a single assignment language ", SIGPLAN Not., 13, 1, pp. 25-33 (Jan. 1978). (Apr.-May 1968).

(11) Dennis, J. B., et al : " Data-flow schemas ", Project MAC, M. I. T., Cambridge, Mass., pp. 187-216 (July 1972).

(12) Kosinski, P.R. : " A data-flow language for operating system programming ", SIGPLAN Not., 8, 9, pp. 89-94 (Sept. 1973).

(13) Rumbaugh, J.:" A data-flow multiprocessor ", IEEE Trans. Computer, C-26, 2, pp. 138-146 (Feb. 1977).

(14) Dennis, J.B. : " The varieties of data-flow computers ", Proc. 1st Int'l Conf. on Distributed Computing Systems, pp. 430-439(Oct. 1979).

(15) Boakelheide, K. : " A high level, graphical, data-driven language ", Data-flow workshop Toulouse (1979).

(16) Davis, A.L. : " DDN's—a low-level program schema for fully distributed systems ", Data-flow workshop Toulouse (1979).

(17) Ruggiero, W., Estrin, G., Fenchel, R., Razouk, R., Schwabe, D. and Vermon, M. : " Analysis of data-flow models using the SARA graph model of behavior ", Proc. NCC, 48, pp. 975-988 (June 1979).

(18) Dennis, J.B. : " Data-flow supercomputers ", Computer, 13, 11, pp. 48-56 (Nov. 1980).

(19) Arvind and Kathail, V. : " A multi-processor data-flow machine that supports generalized procedures ", Proc. 8th Ann. Symp. on Computer Architecture, pp. 291-302 (May 1981).

(20) McGraw, J.R. : " The VAL language : Description and analysis ", ACM Trans. Programming Language and Systems, 4, 1, pp. 44-82 (Jan. 1982).

(21) Agerwala, T. and Arvind : " Data-flow systems ", Computer, 15, 11, pp. 10-13 (Feb. 1982).

(22) Ackerman, W.B. : " Data-flow language ", Computer, 15, 2, pp. 15-25 (Feb. 1982).

(23) Davis, A.L. and Keller, R.M. : " Data-flow program graphs ", Computer, 15, 2, pp. 26-41 (Feb. 1982).

(24) Arvind and Gostelow, K.P. : " The U-interpreter ", Computer, 15,

11, pp. 42-49 (Feb. 1982).

(25) Watson, I. and Gurd, J. : " A practical data-flow computer ", Computer, 15, 2, pp. 51-57 (Feb. 1982).

(26) Backus, J. : " Can programming be liberated from the von neumann style? A functional style and its algebra of programs ", Commun. ACM, 21, 8, pp. 613-641 (Aug. 1978).

(27) 西川，寺田，浅田，磯，国本：" 図的表現によるデータ駆動形並列処理記述言語の一提案" ，昭 56 信学情報・システム全大，595 (1981—10).

(28) 西川，寺田，浅田：" 図的表現によるデータ駆動形並列処理記述言語について" ，信学技報，EC81—74 (1982-02).

(29) Nishikawa, H., Asada, K. and Terada, H. : " A decentralized controlled multi-processor system based on the data-driven scheme ", Proc. 3rd Int'l Conf. on Distributed Computing Systems, pp. 639-644 (Oct. 1982).

(30) 西川，寺田，浅田：" 履歴依存性を許すデータ駆動図式" ，信学論（D），J66-D, 10, pp. 1169-1176 (1983-10).

(31) 西川，浅田，寺田：" データ駆動形実行制御の一方式とその実験的検討"，信学論（D），（投稿中）.

(32) 斉藤：" ＯＳの基礎理論(1)～(3)" ，情報処理，15, 11, 12, 16, 1, (1974-11-1975-01).

(33) 村岡：" 並列処理概論(1)～(3)" ，情報処理，16, 1-3, (1975-01-03).

(34) Bouknight, W. J. : "The ILLIAC Ⅳ system ", Proc. IEEE, 60, 4, pp. 369-379 (Apr. 1972).

(35) 宇都宮：" データフロー計算機" ，bit, 12, 3-9, (1980-03-09).

(36) Gylys, V. B. and Edwards, J. A. : "Optimal partitioning of workload for distributed system ", Digest of Papers COMPCON FALL 76, pp. 353-357 (Sept. 1976).

(37) Stone, H. S. and Bokhari, S. H. : " Control of distributed processes", Computer, 11, 7, pp. 97-106 (July 1978).

(38) Chu, Holloway, Lan and Efe : " Task allocation in distributed data processing ", Computer, 13, 11, pp. 57-69 (Nov. 1980).

(39) Batcher, K. E. : " Design of massively parallel processors ", IEEE Trans. Computer, C-29, 9, pp. 836-840 (Sept. 1980).

(40) Mago, G. A. : " A network of microprocessors to execute reduction languages ", Int'l J. Comput. & Inf. Sci., 8, 5, pp. 349-385 and 8, 6, pp. 435-471 (June 1980).

(41) Kennaway, M. R. and Sleep, M. R. : " Applicative objects as processes ", Proc. 3rd Int'l Conf. on Distributed Computing Systems (Oct. 1982).

(42) Civera, P., Conte, G., Del Corso, D. and Pasero, E. : " The μ* project ", Micro, 1, pp. 38-50 (May 1982).

(43) 甘田，土田，佐藤，後藤，馬場：" 分散処理用計算機のシステムアーキテクチャ ", 情処学会アーキテクチャ研究会資料, 48-3 (1983-03).

(44) Wulf, W. A., Levin, A. and Harbison, S. P. : " Hydra／C.mmp : An Experimental Computer System ", McGraw-Hill (1981).

(45) Swan, D. J., Fuller, S. H. and Siewiorek, D. P. : " Cm*-A modular multi-micro processor system ", Proc. NCC, 46, pp. 637-644 (June 1977)

(46) Gehringer, E. F., Jones, A. K. and Segall, Z. Z. : " The Cm* testbed ", Computer, 15, 10, pp. 40-53 (Oct. 1982).

(47) Arvind and Iannucci, R. A. : " A critique of multiprocessing von neumann style ", Proc. 10th Ann. Symp. on Computer Architecture, pp. 426-436 (June 1983).

(48) 佐藤：" プログラミング用ドキュメンテーション ", 情報処理, 22, 5, pp. 383-389 (1981-05).

(49) 二村，川合，堀越，堤：" ＰＡＤ (Program Analysis Diagram) によるプログラムの設計および作成 ", 情処学会論文誌, 21, 4, pp. 259-267 (1980-04).

(50) " 民生・家電用高機能素子に関する調査研究報告書 ", 電振協 (1983-03).

(51) Zohar Manna (五十嵐訳) : " プログラムの理論 ", 日本コンピュータ

協会 (1975).

(52) Dijkstra, E.W., et al : " Structured programming ", Academic Press (1972).

(53) Bohm, C. and Jacapini, G. : " Flow diagrams, turing machines and languages with only two formation rules ", Commun. ACM, 9, 5, pp. 366-371 (May 1966).

(54) Peterson, J.L. : " Petri net theory and the modeling of systems ", Prentice-Hall (1981).

(55) Arvind, Gostelow, K.P. and Plouffe, W. : " An asynchronous programming language and computing machine ", TR. 114A, Univ. of CA., Irvine (Sept. 1978).

(56) Bic, L., Hartmann, R.L. and Todhunter, J. : " The active graph database machine ", Proc. 3rd Int'l Conf. on Distributed Computing Systems, pp. 178-186 (Oct. 1982).

(57) 寺田, 浅田, 西川, 国本, 矢野 : " PMA 上のデータ駆動形マルチプロセッサシステムについて ", 昭57信学総全大, S1-13. (1982-03).

(58) Gostelow, K.P. and Thomas, R.E. : " A view of dataflow ", Proc. NCC, 48, pp. 629-636 (June 1979).

(59) 寺田, 浅田 : " Passive memoryless architecture ", 情処学会アーキテクチャ研究会資料, 31-2 (1978-06).

(60) 寺田 : " Passive memoryless architecture の交換制御への応用 ", 信学技報, SE78-59 (1978-08).

(61) 浅田, 小谷, 寺田 : " Passive memoryless architecture の一実現法 ", 信学技報, EC79-76 (1979-02).

(62) Terada, H., Grandjean, B. and Pitie, J.M. : " A control scheme for resource allocation in multi-micro-processor switching systems ", Proc. ISS, pp. 1196-1202 (May 1979).

(63) 浅田, 寺田, 白野, 渡辺, 磯, 国本, 西川 : " Passive memoryless

architecture 上のデータフロー形実行制御”，信学技報，EC80-49 (1980-12).

(64) 矢野，西川，浅田，寺田：“データ駆動形 マルチプロセッサシステムの一実行制御方式”，信学技報，EC82-37 (1982-07).

(65) 西川，寺田，浅田，坂口：“図的表現によるデータ駆動形並列処理記述言語の処理系の一検討”，昭57信学総全大，1483 (1982-03).

(66) 寺田，浅田，西川，国本，磯，矢野，坂口：“ＰＭＡによるデータ駆動形マルチプロセッサシステム”，情処学会マイクロコンピュータ研究会資料，18-2 (1981-09).

(67) 磯，寺田，浅田，西川：“ＰＭＡ上のデータ駆動形マルチプロセッサシステムの機能割当に関する一検討”，昭57信学総全大，1415 (1982-03).

(68) 米沢：“ ACTOR 理論について”，情報処理，20，7，pp. 580-589 (1979-07).

(69) “プログラミング言語の最近の動向”，情報処理，22，6，(1981-06).

(70) “非手続き型プログラミングのための計算モデル”，情報処理，24，2 (1983-02).