技報

UNIVAC TECHNOLOGY REVIEW

1981年2月　第0号

日本ユニバック

EDITORS' NOTE

一般的に技術とは何か，と問うならば，それは人間の営む創造と生産のいろいろな局面での人間の社会的な判断力であるといえよう．歴史をひもとくならば，技術は社会に依存して，社会の占める時と所に応じて多様な技術が開花し，ときには破行し，ときには発展してきた．そこで，かりに，人間の時と所を越える自由な構想力を一方の極とし，時と所に従った社会と技術のノルムを他方の極に置くならば，個々の実践技術はこの対立する両極の調和点の追求から生れると考えられる．人間の心の自由な構想は物理的時空を越え，歴史や社会を越えて広がる．一方，この人間の構想が，1つの技術を産むためには，物理的時空の枠の中で客体化され，さらに，それぞれの歴史と社会の枠の中で実現されなければならない．科学という物理的時空の束縛条件や，個別の社会と技術のノルムの束縛条件と，人間の構想の自由とを調和する人間の力が技術力ともいえようか．

私企業は営利活動を行う社会的存在であり，その技術は，企業の占める時と所に応じてそれぞれその調和点を追求する．だから，それぞれの企業における技術もおのずから相異を示すであろう．また，当然，政府や大学といった非営利的な機関における技術の追求とも色彩を異にする．企業体の社会に占める位置や企業規模に応じて，私企業の技術は，自己の浮沈に関する最適解を得るために，自己に“適正な技術”の調和点を追求する．それは，大規模な技術開発で，技術の現状を一変させるようなものではないかもしれない．また，先鋭的な研究開発で，技術のノルムを一変させるようなものではないかもしれない．むしろ多くは，よくアセスメントを行って，個々の技術の実現可能性を自己の時と所に照らして証明し，評価を加えながら，企業“それなりの”実践技術を追求している．時間のスパンの中ではたとえ先鋭的でない技術であっても，自己に適正で，企業体の浮沈を賭けることができると評価でき，また選択できる技術が，私企業の実践する技術である．

この技術の評価と選択のフィルタ，いわゆる技術の徹底したアセスメントそれ自体が，企業体が生きていくための技術であろう．

さて，H. C. Gyllstrom らの**言語処理系の生産技術 UCS** は，ソフトウェアの移植技術を実際に企業体において言語処理系の生産技術として系統的に適用した結果を述べる．同様の言語系と機械系の系統的なとらえ方は，すでに1950年代の UNCOL にみられる．しかし，UNCOL は成功しなかった．また個別のソフトウェアの移植技術は，1964年に Cambridge 大学の M. V. Wilkes 教授が LISP の移植実験を行ったのをはじめとして，W. M. Waite, M. Richard, P. J. Brown 教授らによって多数の実験が大学で行われている．日本では東大の和田英一教授らによって PASCAL の移植実験が行われた．そういう意味では，個別にみると決して新しいものではない．1969年，ローマで NATO の科学委員会をスポンサとするソフトウェア工学の国際会議では，W. S. Brown 氏が“ソフトウェアの移植性”についてのワーキング・ペーパーを提出した．その頃から，ソフトウェア工学上の技術の観点で，ソフトウェアの移植は論議されている．UCS の意義は，効率の最適化問題を評価しながら，実践的な生産技術として実現した点にみられよう．F. W. Stodola の**システム記述言語 PLUS** は UCS を支援している中間言語系の1つであり，古典的な JOVIAL 等を原型としている．システム記述言語としては，現在，システム・ソフトウェア全般を実現するのに使われている．

H. C. Heacox の**ソフトウェア開発言語 RDL** は，Requirement and Development Language の命名から知られるように要求仕様の記 （表紙3に続く）

創刊によせて

富田和夫

周知のように，コンピュータは急速に社会に浸透し，あらゆる分野で広範に利用されている．現代社会の人間とコンピュータのつながりはますます緊密の度を深め，大きな社会的影響を与えつつある．

このコンピュータの広範な利用と急速に進歩したエレクトロニクス技術の関連をみるならば，時を得て，後者が前者の有効な工学的手段となりえた点を見過すことはできない．

計算を機械によって行う構想は古くから人の心に画かれていた．L. da Vinci はすでに10進の加算器を設計している．しかし，彼はその構想を実現するための工学手段として歯車の回転運動を利用したが，実用化には失敗した．また，記号処理によって論理的思考を機械的に行う構想を実現するために，G. W. Leibniz は汎用の推論機械 (Machina Ratiocinatrix) を製作した．彼の二元算術 (Arithmetica Dyadica) や記号結合術 (Art Combinatoria) 自体は，すでに2進のディジタル原理や記号処理の原理を内包していた．しかし，彼が使用できる工学的手段はやはり歯車の機械工学でしかなかった．C. Babbage の計算機においてもまた，歯車の機械的運動はその構想を実現する有効な工学的手段とはなりえなかった．同様に，電気的かつ機械的な素子であるリレーは H. H. Aiken の Harvard Mark I を実現させる工学的手段とはなったが，その後のコンピュータの発展にとって十分なものとはなりえなかった．

結局，エレクトロニクスを手段とした最初のコンピュータは，J. W. Mauchly と J. P. Eckert らによって，1945年に至って ENIAC として完成したのである．同年，J. von Neumann は，A. M. Turing の思考上の万能計算機 (1936年) の動作原理を基本とした，いわゆる今日の von Neumann 型コンピュータの開発を提案しているが，エレクトロニクス技術によるその実現はさらに後年のことである．

今日の半導体工学とその高集積化技術の発達により，演算と記憶のための半導体素子はきわだって改良された．そして，コンピュータの重要な特長である計算速度は著しく高められ，情報の記憶機能は量的に大幅に改善された．さらに記憶能力の量的改善の副産物として，データベース等の情報処理技術の道を開いてきている．一方，「マイコン・ブーム」という流行語からもうかがうことができるように，ハードウェアのコスト・ダウンによって，その用途の多様化と拡大は著しい．

コンピュータの高速化と記憶量の増大は，現在も，エレクトロニクスを工学的手段として続けられている．たとえば，半導体工学では Josephson 素子等の窮極的な動作特性を持つ素子の研究開発が推進されており，一方，磁性の研究を基礎に大容量記憶装置の開発が推進されている．

これらのエレクトロニクスの技術開発と相俟って，今までサイエンス・フィクションで画かれて

きた幾つかのアイデアの工学的な実現の可能性が生まれてきている．すなわち，自然言語処理，パターン処理，非 Neumann 機構などの諸構想を工学的に実現しようとする高度化計画が立案され，これに基づいて，昨今，一層の研究開発が推進されている．

このような多面的な機能拡大を含め，全般にわたってコンピュータをめぐる諸技術は過去に類例を見ないほど急速で広範な変革を遂げようとしている．現代社会は，たしかに上述の単なる高速の計算能力の拡大から一歩踏み出して，多面的な情報処理構想の工学的実現を求めている．エレクトロニクスが，コンピュータの量的改善に十分に有効な工学的手段となってきたように，この多様な情報処理の構想を実現するために，現代の諸工学的手段が時を得て十分有効に働き，さらにその変革を促していくであろう．

コンピュータの機構，いわゆるハードウェアの側面を見ると，コンピュータは現代社会の広範な領域に存在する多様な情報を処理する一般情報処理機械としてさらにその利用が拡大されようとしている．個々の利用分野から見るならば，もはや，コンピュータと呼ぶことは妥当でないとも考えられ，一般情報処理系として現代社会で広く機能していると言っても過言ではない．人間の計算と推論の量的な機能拡大にとどまらず，質的な拡大をめざしている．すなわち，視聴覚系を主とする人間の感覚系や運動系の情報処理機能を拡大する周辺機器とその情報処理技術の研究開発が進められている．このパターン情報処理の技術開発は，現代社会の人間とコンピュータの共生関係，人間・機械系の改善に大きく役立つであろう．また，コンピュータは広く従来の機械の運動系の動作を制御する役割を果たそうとしており，自動車や家庭用の機器でさえもコンピュータ制御されつつある．

このように，コンピュータに関連する技術は，従来の利用を越えて，人間とコンピュータの多様な共生関係を産みながら，同時にその使いやすさを追求している．もはや，単にコンピュータというのが妥当ではないと述べたゆえんである．コンピュータ技術は，「人間と機械の系における情報処理の機能を拡大する」構想を工学的に実現するものと言える．

こうして，コンピュータは骨の折れる単調な頭脳労働から人間を解放し，われわれは真の情報化時代に足を踏み入れたのである．コンピュータによる単純な頭脳労働からの解放は，より自由な創造的思考を促し，同時に人間の諸活動の有効性をさらに拡大させるだろう．いいかえれば，この時代では人は頭脳的に怠け者ではなく，これまでよりも厳しい創造的思考を求められよう．

本誌では，コンピュータや情報処理に関連する技術について，その歴史的な意味や位置づけを明らかにしながら，技術的な問題解決への努力や成果を紹介するのが目的である．また，技術の進歩はそれ自体の中に目的があるわけではない．その歩みは，人類の将来，倫理，生活といった精神的・物質的水準によってのみ正当化される．この意味から，この技術分野の社会的な意義や影響についても，取り扱いたいと思う．

（日本ユニバック株式会社　常務取締役）

言語処理系の生産技術 UCS

The Universal Compiling System

H. C. Gyllstrom, R. C. Knippel
L. C. Ragland, K. E. Spackman

要 約 万能コンパイル系(Universal Compiling System, 略して UCS)は，ある言語の集合と処理機械の集合に対する統一的なコンパイル系の1つである．UCSが持っている一貫したコンパイルの実現方法によって，開発と保守の費用を低減し，信頼性を向上させることができる．UCS 設計の鍵は，原始言語に依存する処理と処理機械に依存する処理とを完全に切り離すことである．この分離は言語と処理機械の双方に依存しない中間言語と辞書(記号テーブル)とを使用することによって可能となる．

Abstract The Universal Compiling System (UCS) is a unified compiling system for a set of languages and architectures. The integrated implementation discipline of UCS results in reduced development and maintenance costs and increased reliability and efficiency. The key to the UCS design is the strict separation of the source language dependent processes from the architecture dependent processes. This is made possible through the use of an intermediate text and dictionary (symbol table) which is both language independent and architecture independent.

1. はじめに

1970年代の初頭に Sperry Univac 社では，コンパイラ開発の分野で新しい重大な事業を行わなければならなくなった．それは，新しいハードウェアやオペレーティング・システムからの要求，新しい標準規格の制定，そして，新しく実用化されたコンパイル技法の採用などにより生じたもので，既存の言語プロセッサの大幅な改良であり，新しい言語プロセッサ——たとえば PL/I や Sperry Univac 社の社内向けの開発用高水準言語である PLUS——の開発である．

Sperry Univac 社では，これらのソフトウェアが互換性のあるコンパイラ体系となるよう，これらすべてのコンパイラを1つのプロジェクト内で開発することになった．その結果，異なる言語プロセッサ間での共通性や互換性を強調した開発思想のもとで，一連のコンパイラが並行して開発された．この開発思想をコンパイラ・モデルと呼ぶ．このコンパイラ・モデルに従って開発された主力商品には

- PLUS——Programming Language for UNIVAC Systems
- ASCII FORTRAN
- PL/I

の3つがある．

UCS (万能コンパイル系) は，あるプログラム言語の集合と処理機械の集合に対する統一的なコンパイル系の1つである．コンパイラ設計に対して，このような包括的な処理方法を行うことによって，次のことが可能となる．それは，1つの言語についていえば，いろいろな処理機械で一様に利用できるということであり，1台の処理機械についていえば，いろいろな言語を一様に使用できるということである．したがって，各処理機械は同一の

Reprinted from ACM SIGPLAN NOTICES, (December 1979), Vol. 14, No. 12, pp. 64-70.

言語の集合を持つことになるばかりでなく，同一の言語定義を持つことになる．この言語の集合は，BASIC, COBOL, FORTRAN, PL/I および PLUS（ユニバック・システムに対する1つのシステム記述言語）を含んでいる．処理機械に対する一様性によって，使用者プログラムの移植性（portability）が保証されるから，原始プログラムを変換しないで1台の処理機械から他の処理機械に移すことができる．また，いろいろな言語間での一貫性（consistency）によって，使用者は応用にもっともよく合う言語が選択でき，また1つのプロジェクト内で複数の言語系を使用することができる．

それぞれのプログラム言語と処理機械の組合せに対して個別のコンパイラをつくる方法が，コンパイラを製作するときの伝統的なアプローチの方法である．その際，コンパイラ中の個別の部分処理の配置は便宜的である．その結果，コンパイラの原始言語と処理機械とに対する依存性はコンパイラ全体に散らばる．これらの依存性は原始プログラムの中間表現として，もっとも単純なもの以外のすべてのコンパイラで使われる，中間言語と辞書にも影響を与えることとなる．伝統的コンパイラのほとんどすべての成分は言語や処理機械の双方へ依存性を持つ．したがって，コードや算法を1つのコンパイラから他のコンパイラへほとんど移すことができない．このような事情によって，新しいコンパイラの開発に要する時間が長くなり，費用もかさんでくる．さらに，コンパイラを個別に開発すると，言語の側から見るといろいろな処理機械間で1つの言語に対する不統一が起こるし，処理機械の側から考えるといろいろな言語間で1台の処理機械に対する不統一が生じることになる．

UCS の実現はこれらの問題の解決を志向している．その第1の目標は万能性（universality）である．この万能性とは，単一のコンパイル系により言語の集合が一様に処理されることを意味する．コンパイル過程で実行する多くの機能はコンパイルする言語と独立であり，コンパイル可能な言語間で共通に利用できる．したがって，この第1の目標は十分実現可能な目標である．このことは異なる言語に対するコンパイラが単一の構造を持ち，そのコンパイル過程の同一の場所で同一の論理機能を実行するということを意味している．このコードと算法の共通性によって，言語間の一様性が得られるとともに，開発の保守の費用を節減することができる．

第2の目標は移植性であって，機械機構に独立なコンパイル処理を固定して，分離することである．コードの移植性により，新しい処理機械へ UCS を移植するための費用および開発期間を軽減することができる．

UCS の第3の目標は，既存の Sperry Univac 社のコンパイラとの上方の互換性(upward compatibility) である．新しくつくられたコンパイラを使うためには，連結編集したプログラム全体を再コンパイルしなければならない，という基本的な問題が生じる．ところが，UCS でコンパイルしたプログラムは再配置可能エレメントのレベルで互換性があるから，現在のコンパイラによるプログラムと UCS でコンパイルしたプログラムを混合して連結編集することができる．したがって，現在のコンパイラから UCS へゆるやかに移行することができる．この性質によって，使用者はシステムの部分集合を常に増加させながらプログラムを再コンパイルし，テストすることができる．

2. UCS の構造

図1に示したように，UCS 設計の鍵は，原始言語に依存する処理と処理機械に依存する処理とを完全に切り離すことである．この分離によって，万能性と移植性とが実現可能

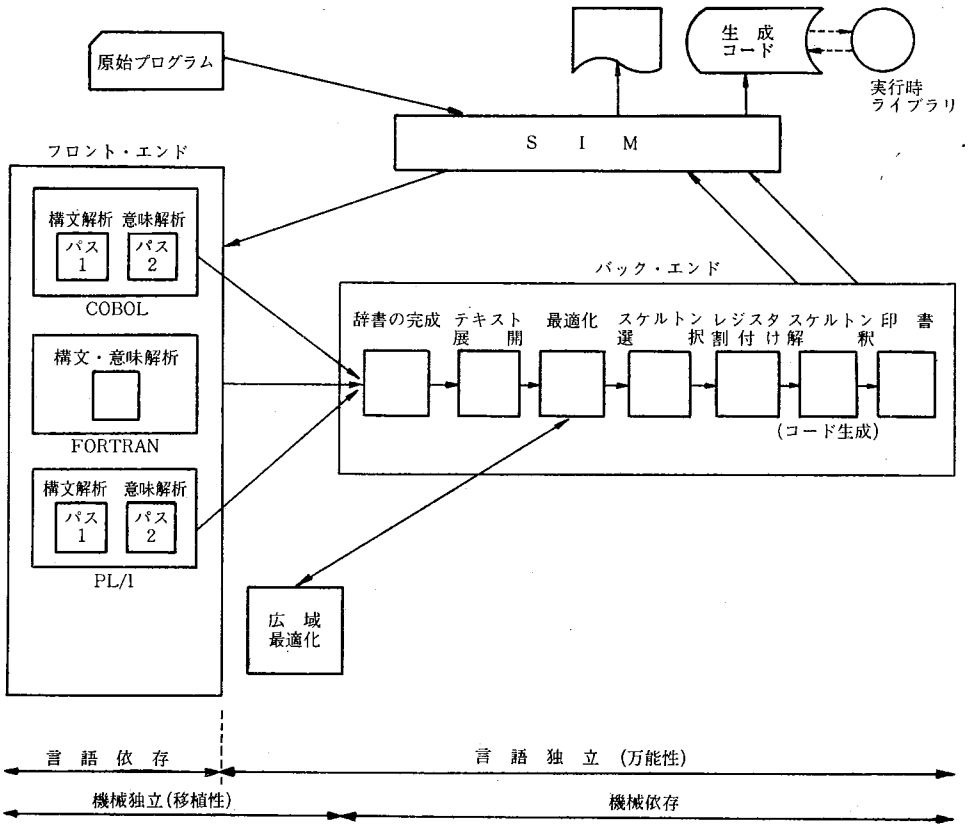


図 1 UCS の構造
Fig. 1 UCS structure

となる．UCS の各原始言語は個別のフロント・エンド (front end) を持ち，各フロント・エンドはその言語に固有の処理を含んでいる．フロント・エンドは完全に処理機械から独立している．一方，UCS のバック・エンド (back end) 部分は処理機械に依存する処理を行い，完全に原始言語から独立している．各処理機械は別個のバック・エンドを持つ．言語依存処理を機械依存処理から分離するために，言語と機械との双方に独立な 1 つの共通な中間言語と辞書が導入される．フロント・エンドは原始プログラムをこの共通な中間言語と辞書を使う表現に変換する．バック・エンドはこの表現を対象の処理機械で使われる目的コードに変換する．共通中間言語と辞書とが機械から独立しているということと，高水準のシステム記述言語を使っていることによって，任意のフロント・エンドは移植可能となる．また，中間言語と辞書を原始言語から独立させることによって，バック・エンドは万能 (universal) となる．すなわち，すべての原始言語に適用できるようにする．

図 1 に示した UCS の構造はフロント・エンドとバック・エンドからなり，これらの 2 つの部分はそれぞれいくつかのフェーズに分割される．フェーズという用語はここでは分離された論理処理の 1 つを示すのに使う．

2.1 フロント・エンド

第 1 フェーズ，すなわち構文解析では，記号列で表現された原始プログラムをプログラム解析に向く形式にコード化した中間テキストと辞書とへ変換する．一般に構文解析でつくられた中間テキストは，フロント・エンドで最終的に作成される共通の中間テキストと

は異なるであろう．構文解析での中間テキストの形式は言語の複雑度によって決定され，たまたまフロント・エンドで作成される共通のテキスト形式とは異なるかもしれない．このパスの間に，識別子と定数の入れ場所をつくりだすことで，辞書の製作を開始する．この時点で共通の辞書に入るデータ型は原始言語のデータ型である．構文解析には主として1つの主要ツール GSA (General Syntax Analyser, 一般構文解析プログラム) が使用される．GSA はすべてのフロント・エンドで使われる．

第2フェーズは意味解析であり，前のフェーズで構文解析された原始言語の構成の意味を決定する．このフェーズでは原始言語の特異性を解消し，言語に固有な構成は共通の構成に変換される．また，意味解析では辞書が決定したもののうち，機械に独立した部分の処理も行われる．これには，属性の決定，省略時解釈の適用，および言語のデータ型を一般データ型へ変換することを含んでいる．一般データ型とは，共通の辞書の中で使う言語にも，処理機械にも独立なデータ型のことである．

フロント・エンドの移植性を保証するために，フロント・エンドの外部インタフェースはシステム・インタフェース・モジュール (System Interface Module, 略して SIM) と呼ぶルーチンの集合に分離しておく．SIM は一般化したシステム・インタフェースを用意する．これによって，フロント・エンドが機械独立な環境で動作できるようになる．たとえば，原始行の読込み法を変えるときには，フロント・エンドを変更するのではなく，SIM ルーチンを変更する．

構文解析や意味解析のフェーズで行われなければならない処理量は，いろいろな言語の間でかなりのばらつきがある．PL/I や1982年(予測)米国標準規格の COBOL のような言語では意味解析においてかなり多くの処理が必要であるから，構文解析と意味解析に分けた別々のパスを必要とする．ALGOL や FORTRAN のような比較的単純な言語では，パスを分ける必要がない．

2.2 バック・エンド

バック・エンドにおける第1フェーズは辞書を完成させるためのものである．すなわち，このフェーズでは，まず処理機械に依存する情報を共通の辞書に付け加えていく．UCS 内で1つの分離したフェーズとして，辞書の完成というフェーズを置くのは，言語と機械への依存性を分離するための直接的結果である．伝統的コンパイラにおいては，ここで行う機能は通常，意味解析の中に組み込まれる．辞書完成フェーズは記憶域のサイズを決定し，相対番地を割り付け，一般データ型を機械のデータ型へ変換する．

次のバック・エンドのフェーズ，すなわちテキスト展開では，共通の中間テキストの列を機械に依存するテキスト列へと展開する．辞書の完成の場合のように，テキスト展開は1つの分離したフェーズをなすが，それは言語と機械に依存する部分を分離したためである．このフェーズはほとんどの伝統的コンパイラにおいては意味解析の一部に入れられている．テキスト展開は，また，最適化処理のための前処理ともいえる．高レベルのテキストは低いレベルのテキストに展開して，より多くの最適化操作が行えるようにする．これにはすべての番地付け計算や機械に依存する型の変換が含まれる．

最適化では，プログラムの共通中間テキストと辞書表現を等価な効率のよいプログラムに写像する．最適化のフェーズには2つの大きな柱がある．1つは機械に依存する最適化であり，もう1つは機械に独立な広域最適化である．機械に依存する最適化には，対象とする機械の持つ命令群の利点を最大限生かすための最適化が含まれる．広域最適化は機械にも言語にも独立な一般的な最適化を行う．したがって，同一の広域最適化処理がすべての原

始言語に適用でき，無修正で処理機械から処理機械へ移すことができる．これは，展開された中間テキストが機械や言語に依存したテキスト・エントリを使用していないから可能なのである．テキスト展開後は機械や言語に依存しているのはエントリそのものではなく，エントリの組合せ方(順序)である．広域最適化は個々のテキスト・エントリについてその変換を行うから，テキスト・エントリの順序が変わっても処理に影響を与えない．この最適化にはプログラムのフロー解析を行って，プログラム全体を通して最適化変換を行うという意味で広域的である．これらの変換には折たたみ (folding)，定数式の評価，共通式の除去，コードの移動，および演算子強度の削減 (strength reduction) が含まれる．

スケルトンとは，中間テキストの個々のエントリからどんな目的コードを生成すべきかを示している表のことである．従来，レジスタ割付けという処理はコンパイラ処理の中でも複雑な処理の1つだったが，UCS ではレジスタ割付けを簡単にする画期的方法を取り入れている．すなわち，目的コード生成処理をスケルトン選択とスケルトン解釈の2つに分け，その間でレジスタ割付けを行う方法である．

実際の目的コードはレジスタ割付けの後のスケルトン解釈のフェーズで生成される．レジスタの割付けのフェーズは，基本モジュールと最適モジュールの2種類の呼出し方が可能である．基本レジスタ割付けとは基本ブロック，すなわち，レーベルや分岐間の直線的テキスト列だけを見てレジスタ割付けを行うことである．最適レジスタ割付けでは最適化でつくられたフロー解析の結果情報を使い，よりよいレジスタ割付けに対するインテリジェントな処理を行う．

バック・エンドの次のフェーズはスケルトン解釈である．目的モジュール生成ルーチンは最終的な辞書と中間テキストとを目的コードに変換する．目的コードは，ライブラリと結び付けられると，目的プログラムとなる．

バック・エンドの最終フェーズは印書で相互参照表や属性の印書いわゆる L オプション・リストと呼ばれる生成コードの印書などを行う．

UCS の実行時ライブラリは，すべての UCS の原始言語に対して実行時支援を行う．実行時ライブラリは1つのルーチンの集合で，共通の実行時のデータ構造を使い，共通の入出力パッケージを含む他のシステム要素と一様なインタフェースを持っている．エラー処理のようなある分野では，実行時ルーチンは実行するプログラムの言語に敏感でなければならない．しかし，実行時支援を要する多くの特性はいろいろな言語で共通であり，ライブラリのコードは可能なかぎり言語間で共通に利用できる．実行時環境は基本的には各言語で同じであり，生成されたコードからのインタフェースは言語に依存しない．したがって，言語間の呼出し (calls) によるオーバーヘッドは本質的に縮められる．また，言語に独立な使用者インタフェースを持つ広範なデバッギング用のツールが，UCS の実行時支援ルーチンに含まれている．

3. 万能性と移植性

上述のように，UCS には2つの重要な特性あるいは次元がある．それは万能性と移植性である．図1は，万能性と移植性のそれぞれの次元からの異なった観点でながめることができる．第1の次元すなわち万能性の観点からは，UCS は多くの言語をコンパイルできる単一のシステムである．コンパイル可能な各言語で書かれたプログラムは，対応するフロント・エンドによって，共通の中間テキストと辞書に変換される．原始プログラムのこの中間表現は，与えられた処理機械に対する共通のバック・エンドへの入力となる．したが

って，すべての言語依存処理をこのバック・エンドから消去することによって万能性が可能である．

第2の次元，すなわち移植性は機械依存処理をフロント・エンドから除くことによって実現される．これによって，フロント・エンドは異なる処理機械間で移植可能となる．共通の広域最適化プログラムも，また移植可能である．1つの共通なソースがこれらモジュールに存在する．しかし，バック・エンドは，コードのかなりの部分を同類の機械に移行できるが，それぞれの処理機械に固有である．UCS では，ある種の移植性を持たせるために，1つの高水準のシステム記述言語を使用している．このシステム記述言語は PLUS と呼ばれ，システム・ソフトウェア生産のために Sperry Univac 社がつくった高水準言語である．

4. UCS の利点

製作者側から見た UCS の利点は自明である．たとえば，新しい言語や新しい機能の追加に対して，低い初期開発費用，高い保守性，さらに短期間で安価な開発が一般に実現できる点である．

UCS 自体の初期開発費用は，各言語に対して別個にコンパイラを開発する費用の和よりかなり低い．一度 UCS を製作してしまうと，新しいバック・エンドと実行時システムをつくれば，言語の集合を新しい機械に移植することができる．

同様に，UCS の保守費用は，UCS でつくることのできる言語全体と同じ範囲内で伝統的コンパイラを支援するのに要する費用（現在値）よりもかなり低い．UCS を実現したすべての機械で保守しなければならない部分は，フロント・エンドの集合と1つの広域最適化プログラムだけである．また，各機械については，UCS のすべての原始言語を支援するには1つのバック・エンドと1つの実行時ライブラリがあればよい．

UCS の言語集合に新しい原始言語を加えたり，既存の言語に新機能を追加することが容易にできる．一般に，1つの新しい言語を開発するのに3年以内ですみ，またコンパイラの製作前に要する投資のうちかなりの部分が不要となる．現在の UCS に新しい言語を加えることは，新しく別個のコンパイラを製作する費用の数分の1の短い期間で実行できる．もともと UCS の実現は多くの異なる言語に適応するように設計したので，既存の言語に新しい特性を追加するには，追加する機能がすでに他の言語で実現されている場合が多いことを利用すれば，一般に低い費用で可能であろう．

さて，使用者側の利点はどのような点であろうか．安定したコンパイラであるとか，システム費用が安くすむ，などという製作者側の利点から導かれるものにすぎないのかという疑問が生じよう．また UCS システムが複雑であることによって，その分だけ効率が落ち，よくないコンパイラやコンパイルされたよくないプログラムができて，初めのうちは使用者にとって利点がないように見えるかもしれない．しかし，実はそうではない．第1に効率の損失はなく，第2に使用者はいろいろな面で UCS から直接に利益が得られる．

万能性と移植性との両方を満足させている UCS コンパイラは，特定の単一の処理機械と特定の言語につくり付けられた個別の言語コンパイラより，コンパイル速度が遅いように思われる．しかし，UNIVAC シリーズ 1100 のコンパイラ製作の経験から，原始言語に関係なく，コンパイル過程は同一の基本機能に分割しても効率のよいコンパイラを構成できることがわかった．一方，UCS への万能性の要求が，UCS 中で従来解決されていなかった言語に独立な部分へ，新たにある種の複雑度を持ち込むことになるという．しか

し，この複雑度の問題は論理機能を，たとえば辞書の完成とかテキスト展開のような，追加したフェーズに分割することによってより対処しやすくすることができる．フェーズを分割するということは必要な記憶域の大きさが小さくなるという長所も持っている．UNIVAC シリーズ 1100 において，UCS が必要とする主記憶域は現在使用中のコンパイラよりも少ない記憶域ですんでいる．またこの記憶域要求の減少により，UCS を Sperry Univac 社の規模の小さい処理機械にも適合させることができた．コンパイル速度に対する UCS の複雑度による影響を最小化するために，コードの選択的起動法を利用している．UCS のバック・エンドはいまあるすべての言語機能を支援することができる．しかし，そのうちで費用のかかる機能が原始プログラムにおいて使われない場合には支援するコードを実行しないようにし，必要なときだけに実行できるようにコンパイラを構成する．たとえば，PL/I における構造体の可変長配列を処理するコードはいわゆるミニ・パスという方法を使って実現され，決して FORTRAN のコンパイル中には起動されない．したがって，FORTRAN のコンパイル速度はこの言語機能を含めても遅くなることはないだろう．

UCS の万能性に対する要求は生成されたコードの効率を悪くするように思われるが，実際にはまったく逆で効率はよくなる．なぜならば，UCS は原始言語のすべてを支援しなければならないという要請によって，最適化プログラムとバック・エンドには技術の粋が集められ，より多くの“技法”と“特殊ケース”の処理も数多く集められているからである．そして，これらの技法は任意の言語で書かれた任意のプログラムに使用できるのである．それゆえ，たとえば，PL/I の開発で蓄積され，最適化やコード生成問題に対する解として導入された算法を COBOL で書いたプログラムに利用することができ，よりよいコード生成を行うことができる．同様に，記号列処理を行う FORTRAN や PL/I プログラムは COBOL のために開発した記号列処理の優れた方法から利点を得ることができるからである．

UCS の移植性の要求は，同じ言語定義を持つ同じ言語集合がすべての Sperry Univac 社のシステムで実現されることを意味するから，使用者にとっては直接の利点を得ることになる．システム間の言語の互換性によって，使用者の高水準言語のプログラムは真に移植可能となり，またシステム間の移行を非常に単純化することになる．この言語の互換性が，Sperry Univac 社の異なる機械で構成されるネットワーク内に，進んだ分散処理を実現させる鍵となっている．

UCS での原始言語で書かれたすべてのプログラムに対して実行環境が同一であるから，使用者は UCS の万能性の要求から利点を受けることができる．このことによって，1つの応用とかプログラムとかのうちで複数の言語を使用できるし，またその使用をすすめることになる．言語間の通信は効率的であるから，使用者は実行速度の低下を憂れうることなく用途に応じた最良の言語を選択できる．

最後に，使用者はプログラムを開発するための環境が改善されたという点で UCS の利点を利用できる．現在使用できるデバッキングやエラー・チェックのためのツールは完全なものである．コンパイラの生成した情報は，会話型シンボリック・デバッキング・システムによって実行時に使用されるのみならず，任意の言語で書かれたプログラムのダンプを印書するときにも使用されるであろう．FORTRAN や PL/I のような言語に対するコンパイラのいくつかは過去にある種のデバッキング用のツールを持っていたが，UCSでは開発費用をいくつかの言語に拡散できるから，そうした完全なシステムの実現が UCS で

可能になる．そこで，使用者は総合的なデバッキング機能を得るだけでなく，これらの機能の使用方法がどの言語にも共通であるという重要な点によって，UCS の利点を享受することになる．

5. おわりに

UCS における言語間の互換性の増大，原始言語の移植性，デバッキング機能および費用の節減，システムの安定化等によって，万能コンパイル系は使用者と開発・提供者の双方に利点を与えるようなコンパイラの開発に有意義かつ重要な一歩を演じることになる．さらに，UCS の上方の互換性によって，現在の Sperry Univac 社の利用者は急激な変化をもたらさずに，これらの利点を十分に活用することができる．

(1100 ソフトウェア開発部　森沢 好臣 訳)

執筆者紹介　H. C. ギルストローム (Hans C. Gyllstrom)
1969年米国 Iowa 大学においてコンピュータ・サイエンスで Ph. D. を取得．Sperry Univac 社に入社．現在，COBOL, FORTRAN, PL/I, UCS などのシリーズ 1100 のコンパイラの開発を担当する言語プロセッサ開発部のグループ・マネージャ．Minnesota 大学兼任講師．

編集者注　本稿は昭和54年5月11日に行われた，ユニバック研究会全国会議 (於，大阪商工会議所) 1100 部会での H. C. Gyllstrom の講演「ユニバックにおける言語プロセッサ開発――過去・現在・未来」*SYSTEMS*, No. 145, 5・6 号，1979) の内容をも追加している．

システム記述言語 PLUS

The PLUS Programming Language

F. W. Stodola

要 約 PLUS (Programming Language for UNIVAC Systems) は，Sperry Univac 社で設計された社内のシステム・プログラム開発用の高水準プログラム言語で，これまでにいくつものプロダクト開発に使われてきた．本稿では，PLUS のコンパイル時の諸機能，インライン機能，ASSIGNED 属性について述べる．また，データの抽象的な表現と機械に依存した表現の両方の必要性を満たすために採用した方法についても述べる．

Abstract PLUS, Programming Language for Univac Systems, is a high level language designed by Sperry Univac for use by Sperry Univac systems programmers. It has been used in the development of compilers, interpreters, data management systems, security facilities, and operating systems.

It is not the intent of this paper to give an overview of the PLUS language. Such a discussion would become much too lengthy. Rather, emphasis will be placed upon PLUS's approach to data entities, its INLINE facility, and a selection of its compile-time facilities.

1. はじめに

今日，抽象的プログラミングの概念が強調されている．プログラマは，データの内部表現よりも，データの働きや目的という観点から物事を考えた方がよいといわれる．データの抽象的な性質の例は，次のようなものである．

1) −10 から +100 までの範囲(両端を含む)の整数値を扱える整数型データ
2) 真または偽の条件値を表す条件型データ

抽象的な概念でプログラムを作成することは，システム・プログラミングの分野においても，おおいに有益である．まず，移植性が高まる点である．移植性がある，ない，というはっきりしたものではなく，多かれ少なかれ程度の問題である．アルゴリズムに移植性がある場合には，プログラムの移植性はその抽象性に比例する．完全な移植性のあるプログラムは機械に対して独立である．PLUS は移植性も機械独立性も自動的に保証するわけではないが，言語構成をうまく利用することによって，その実現の可能性が増加する．プログラミングは抽象的に行えるのが理想である．しかしそれが不可能な応用分野も多い．そのような応用分野では，データはたとえば8語からなる入出力パケット(最初の2語は12文字のファイル名を含み，他のフィールドは命令符号，バッファの番地，ドラム上の番地データの語数等を定形式で含む) といった機械の物理的な性質に，より多く依存している．

2. データの表現

PLUS では，抽象的な性質とともに，このような正確なデータの割付けをも可能にしなければならない．PLUS では抽象 (abstract)，詳細 (specific)，機械 (machine) という三段階のデータ指定方法によってこれを具体化している．

Reprinted from ACM SIGPLAN NOTICES, (January 1980), Vol 15, No. 1, pp. 146-155.

2.1 抽象指定

データの抽象的な性質を示す属性として，表 1, 2, 3 がある．PLUS で抽象的なデータを宣言するには，これらの表に述べた属性だけを用いる．データの表現方法については何も仮定されず，データに関する演算は表現方法に依存しない．抽象的なデータ指定の例を以下に示す．

```
DECLARE DAY_OF_MONTH     : INTEGER RANGE(1 TO 31),
        YEAR             : INTEGER(4),
        DAY_OF_WEEK      : STATUS(S'SUNDAY', S'MONDAY',
                                  S'TUESDAY', S'WEDNESDAY'
                                  S'THURSDAY', S'FRIDAY',
                                  S'SATURDAY'),
        ARRAY_OF_REAL(100): REAL,
        TITLE            : CHARACTER(10),
        NAME             : [LAST_NAME  : CHARACTER(12),
                            FIRST_NAME : CHARACTER(6)],
        EMPLOYEE_ENTRY   : BASED
                           [EMPLOYEE_NUMBER : INTEGER(5),
                            NAME : [FIRST_INITIAL : CHARACTER(1),
                                    MIDDLE_INITIAL : 1 CHARACTER,
                                    LAST_NAME      : 12 CHARACTER]];
```

表 1 基本的なデータの型
Table 1 Basic data types

INTEGER：	整　数
REAL：	実　数
LOGICAL：	ビット・パターン
POINTER：	BASED 変数の世代を識別する番地を持つ
CHARACTER：	文字列
STATUS：	状況を名前で表す，数え上げ
CONDITION：	論理値，条件

表 2 記憶領域の類別
Table 2 Storage classification

STATIC：	データに割りつけられた記憶領域が常に確保されていること
BASED：	世代がポインタによって識別されること
AUTOMATIC：	手続き実行中にだけ記憶領域が確保される
EXPORTED：	プログラム単位の外でも使われるデータであること
IMPORTED：	外部で定義されているデータをプログラム単位の中で使うこと
UNALTERABLE：	値を変えてはならないこと

表 3 付加的な属性
Table 3 Qualifying attributes

SIGNED：	符号つき
UNSIGNED：	符号なし
RANGE：	値の範囲を示す

2.2 詳細記述

データの詳細な性質を表すには表 4, 5 の属性を指定する．

詳細なデータ記述は，抽象的に使われるデータの指定に用いてもよい．その場合，この種のデータに対する演算がデータの表現方法に依存するというのではなく，単に宣言中に機械の性質を示す属性を含んでいるというにすぎない．

詳細なデータ記述によって，データの記憶領域の割付け方が指定できる．外部的なプロセッサやオペレーティング・システム自体であらかじめ形式が定められているデータとは，この方法でインタフェースをとることができる．詳細なデータ記述を備えたデータは，対応する抽象的なデータと同じ方法で使用できる．その例を以下に示す．

```
DECLARE YEAR              : INTEGER 18 BITS;
DECLARE CHEBYSHEV_VALUE   : REAL WORD;
DECLARE NAME_OF_FILE      : FIELDATA CHARACTER(12);
DECLARE PACKED_ENTRY      : MAPPED
                          [GENDER : STATUS(S'MALE',S'FEMALE') 6BITS,
                           PROPERTIES : CONDITION BITS(12),
                           INDEX : 18 INTEGER BITS];
```

表 4 記憶領域の単位
Table 4 Storage size

BIT :	寸法をビット数で表す
BYTE :	寸法をバイト数で表す
WORD :	寸法を語数で表す
SHORT WORD :	寸法を 1/2 語単位で表す
LONG WORD :	寸法を 2 語単位で表す
MAPPED :	構造体の記憶領域の割付けが，コンパイラが決定する方法でなく，プログラマが指定したとおりに行われることを示す

表 5 機械表現
Table 5 Machine representation

PACKED DECIMAL :	パック 10 進数
ASCII :	ASCII コード体系
EBCDIC :	EBCDIC コード体系
FIELDATA :	FIELDATA コード体系
ASSIGNED :	データを機械の特別な装置レジスタや特定番地に割りつける

2.3 機 械 指 定

機械に依存したデータの使用法が必要なときには，MACHINE 属性をデータ宣言中で与えなければならない．たとえば算術演算で文字型データを使うとか，整数型のデータの一部分だけを参照するとか，論理型データに文字型データや実数型データを代入するといった場合，明確に文書化しておくためにこの属性が用いられる．PLUS では，この属性を必要とするデータ型と演算との関係に明確な基準があり，違反した場合には診断メッセージが出力される．MACHINE 属性を与えて宣言したデータ例を次に示す．

```
DECLARE ITEM                    : SIGNED INTEGER 36 BITS MACHINE;
DECLARE POINTER_TO_NEXT_NODE
                                : MACHINE POINTER WORD;
DECLARE NAME                    : 8 EBCDIC MACHINE CHARACTERS;
```

この例では，MACHINE 属性により，3 つの変数(ITEM, POINTER_TO_NEXT_NODE, NAME) が機械に依存したデータ表現方法で演算することを示し，このことを明確に文書化している．

2.4 抽象的なデータの移植性

データを抽象的に使うときには，つねに抽象的な属性を用いた宣言が必要となる．たとえば，整数型のデータを機械の性質に関係なく使うのであれば，その大きさは RANGE 属性，あるいは値を表現するのに必要な10進数の桁数を示す精度指定によって指定すべき

である．このことを，次の簡単な例で示そう．

変数が1から500までの範囲の値を持つとする．プログラムが UNIVAC シリーズ1100上でコンパイルされるのであれば，BYTE 属性によって詳細な表現方法が与えられる．シリーズ1100上では1語が36ビットからなるので，その PLUS コンパイラは1バイトにつき9ビット，1語につき4バイトを割りつける．9ビットというのは，もちろんこのデータが取りうる値を表現するのに十分である．このプログラムをUNIVACシリーズ90上に移したとしよう．シリーズ90の1語は32ビットであり，1バイトは8ビットである．シリーズ90 PLUS コンパイラは当然そのように割りつける．そうすると，255より大きい値が表現できず，プログラムは正しく作動しなくなる．ところが，RANGE 属性を用いて，たとえば RANGE(1 TO 500) のように指定して，記憶域の割付けをコンパイラにまかせておけば，どちらのシリーズでも十分な記憶領域が得られる．さらにその記憶領域を簡単にアクセスできる部分語単位に割りつけることもコンパイラが行う．

もっとも BYTE, SHORT WORD, WORD のような詳細記述属性を選び，十分注意すれば，どちらの機械でも正しいコンパイルや実行がなされる．しかし RANGE 属性を用いれば，宣言したデータがとる値の範囲が明文化できるだけでなく，そのデータに対して各機械のもっともアクセスしやすい最小単位の記憶領域が割りつけられる．

3. インライン機能

PLUS のプログラムは，データの参照方法で引き数の受け渡しさえ正しく行うなら，他の言語で書いたプログラムとインタフェースをとることができる．ふつう，システム・プログラムの分野では，オペレーティング・システムとインタフェースをとる必要があり，これをアセンブラ・コードに頼っている．PLUS には，直接この言語で書けない機能については一連の機械語のコードを生成する機能がある．オペレーティング・システムの ER 命令の生成はこの具体例である．

この機能はインライン機能と呼ばれ，コンパイラとインタプリタが用意されている．インライン・コンパイラは独立しており，プログラム中のインライン定義を解釈可能なコードに翻訳する．このコードでインラインのライブラリをつくっておくと，PLUS プログラムから必要に応じて参照できる．インライン・インタプリタは実際にはPLUSコンパイラの一部であり，PLUS プログラム中でインラインを参照していると呼び出される．そして，機械語レベルの命令語またはデータ語をいくつか生成する．インライン参照に対してインタプリタが生成したコードは，PLUS コンパイラがコンパイルした目的コードの中にすべて直接埋め込まれる．

この機能を使えば，ER 命令のような特殊な指令を出すのにアセンブラ・プログラムに頼る必要がなくなる．また手続きの呼出しと復帰，引き数の授受のためのオーバヘッドがないので，オペレーティング・システムの効率をとくに重視するときには有効である．

インラインの参照時に渡す引き数の個数を参照ごとに変えても構わない．このために引き数の個数と型，および個々の引き数の参照形式を判定する機能が備わっている．インライン・インタプリタがPLUSコンパイラの一部であるため，これら必要な情報をコンパイラの情報テーブルから探し出せる．次に述べるのは，シリーズ 1100 上で作動するインライン・プロセッサのいくつかの機能の簡単な紹介である．

組込み関数……引き数に関係する情報を探すために多くの組込み関数が用意されている．これらにより，引き数が定数，単純変数，ラベル，手続き，配列，BASED 変数，部

分参照のいずれであるかを判定する．また，そのデータの大きさと番地内での開始位置が得られる．さらに，引き数の参照，または引き数が BASED 変数の場合には，その一部の参照（たとえばポインタ）がどのレジスタにロードされているかも判定できる．（組込み関数は該当のレジスタの番号を返し，使用によってはインライン中でこれが利用できる．）

名前の参照……インラインの名前そのものは，渡された引き数の個数を表している．インラインの名前に添え字を1つつけると，渡された引き数そのものを参照することになる．いま，インラインの名前が INL であるとすると，　LA, U　A0, INL　は渡された引き数の個数をレジスタ A0 にロードすることになり，　LA　A1, INL(i)　は i 番目の引き数の内容をレジスタ A1 にロードする．もちろん，これに対して生成される実際のコードのオペランド欄は，INL（または INL(i)）に置き換わる値（番地）となる．

インラインの名前に添え字を2つつけた形（INL(i, j)：i と j は整数）の参照は，渡された i 番地の引き数に関連する情報を返す．添え字 j は返される情報の種類を示す．

局所的なラベル……インライン・プロセッサでは局所的なラベルを生成して，そこに飛ぶことができる．このラベルはその特定のインラインによって生成されたコード中だけのものとみなされる．

外部名……インライン定義中にない名前は，未定義の外部名とみなされる．外部名はテーブルに納められ，PLUS コンパイラのコード生成段階で適当な外部参照が生成される．

相等変数……相等変数（アセンブラ言語の EQU 指示語の働き）はコンパイル時だけ存在し，インラインの展開過程の補助として使うことができる．通常は引き数の並びへの添え字とか，局所的なカウンタとして使われる．

診断機能……インラインの中から，誤り，警告，注記の三段階の診断メッセージが出せる．誤りおよび警告は PLUS コンパイラのコンパイル時のメッセージの個数に反映される．

条件つきコード生成……インライン機能では，条件つきあるいは繰返しのコード生成ができるように，ON と OFF の指示語を備えている．ON 指示語に指定した式の値が真であると，ON のブロックが解釈される．偽であると，対応する OFF 指示語の次まで処理の制御が移される．ON のブロックは，入れ子にすることができる．

ESCAPE 指示語……これを直接含む ON ブロックの処理を終了させることができる．また，この指示語に外側の ON ブロック名を指定しておけば，そのブロックとそれに含まれるすべてのブロックの処理を終了させることができる．

```
P     INLINE                .
I     EQU       1           .INITIALIZE LOOP COUNTER
LOOP  ON        P GE I      .CONDITIONAL-DO FOR EACH PARAMETER
      :
      LA        A0,P(I)     .LOAD THE I'TH PARAMETER-FILE
                            .PACKET (INLINE GENERATED CODE)
      ER        PFS$        .PERFORM PROGRAM FILE SEARCH
                            .(INLINE GENERATED CODE)
      :
I     EQU       I+1         .INCREMENT PARAMETER INDEX
      GO        LOOP        .ITERATE
      OFF                   .
      :
      END                   .END INLINE
```

4. ASSIGNED 属性

シリーズ1100の ER 命令では情報自体あるいは情報を詰めたパケットの番地をレジスタに入れて渡すことがよく行われる．同様に情報がレジスタを介して返されることもよくある．この命令を処理するためには，エグゼクティブのプログラム内でそれらのレジスタを直接アクセスしなければならない．PLUS ではこの要求を，プログラム内で変数とある特定のレジスタ，あるいは絶対番地とを関係づけることによって満たしている．これを行うのが ASSIGNED 属性で，必ず MACHINE 属性とともに指定しなければならない．

```
DECLARE VAR_ITEM1 : LOGICAL WORD MACHINE ASSIGNED('A0'),
        VAR_ITEM2 : LOGICAL WORD MACEINE ASSIGNED(042);
```

上記の宣言は，変数 VAR_ITEM1 とレジスタ A0 とを，また変数 VAR_ITEM2 と 042 番地とを関係づけている．このようにすると，ある ER 命令に関する情報がレジスタ A0 を介して渡された場合，割込みを処理するプログラムでは変数 VAR_ITEM1 を参照することによって，その情報を得ることができる．また，042 番地の内容を得たり変更したりするには，変数 VAR_ITEM2 を参照して行えばよい．

5. コンパイル時の機能

原始プログラムと目的プログラムの印書，原始プログラムの走査，あるいは条件つきのコンパイル等の制御は，PLUS のコンパイル時の指示文を用いる．このとき，もっともよく使われる重要な文は，COPY 文，INCLUDE 文，RESUME 文および DEFINE 文である．前もって処理した宣言を後のコンパイル操作のために保存しておく機能として，ENVIRONMENT 機能がある．

5.1 COPY 文

COPY 文は，他の場所にある原始プログラムをこのコンパイルの処理対象に含めるという機能を持っている．いくつかのモジュールに共通した宣言とかコードとかを別のコード単位に集めておくと，それを必要とするモジュールはこのコード単位を複写すればよい．この利点は，変更がそのコード単位だけですむので，保守性が高まるという点である．

COPY 文で指定するのは，複写される原始プログラムの存在場所 (UNIVAC シリーズ 1100 ではエレメント名)を示す文字定数である．複写された行は，COPY 文のあった所に現れ，あたかも元の原始プログラムの一部であるかのように処理される．元の原始プログラム自体は COPY 文では修正されない．COPY 文は再帰的に参照するということはできないが，入れ子にしてもよい．

COPY 文では次の 2 つのオプションが指定できる．NO SOURCE オプションは複写してきた原始プログラムをコンパイル・リスト上に印書しないことを意味する．IMPORTED オプションは，広域的なデータの定義と初期値の設定を 1 つのコピー単位だけで定義することを可能にする(通常，EXPORTED 属性を持つものと IMPORTED 属性を持つものの二組が必要)．このオプションがあるときは，複写されるコードの中にあるすべての EXPORTED 属性が IMPORTED 属性に変更され，初期値の設定ならびに EXPORTED 属性を持つ手続きの本体は無視される．

```
COPY('LIBSOURCE', NO SOURCE);
        "The source code contained in LIBSOURCE will be copied
        "into the program but will not be included into the
        "source listing.
```

```
COPY('FILEA', IMPORTED, NO SOURCE);
                "Assuming FILEA contains the following declaration
                "DECLARE COUNT : INTEGER(4) EXPORTED : =5;
                "it would be processed as if it were
                "DECLARE COUNT : INTEGER(4) IMPORTED;
                "Thus variable COUNT is initialized only in the code
                "unit exporting its declaration.
```

5.2 INCLUDE 文と RESUME 文

INCLUDE 文と RESUME 文は原始プログラムを条件つきでコンパイルすることを可能にする指示文である．INCLUDE 文で指定された条件式が真ならば，INCLUDE 文とそれに対応する RESUME 文の間の原始プログラムがコンパイルされる．偽ならば，対応する RESUME 文の後からコンパイルが続けられる．INCLUDE 文と RESUME文は入れ子にしてもよい．ただし，コンパイルされない部分に含まれる INCLUDE 文と RESUME 文の対は当然コンパイルされない．

```
INCLUDE(COMPILE_FOR_9000);
        COPY('SYSTEM');
        DECLARE ...
  ⋮
RESUME;
```

5.3 DEFINE 文——形式定義

形式定義の第一の目的は，宣言とコードの両方においてプログラムをパラメタ化することである．プログラムを構成するために用いた識別子の意味を適当に定義しさえすれば，そのプログラムを別の環境に合うように簡単に変更することができる．形式定義はある識別子が定数とか一群の属性の列とかを表すように，定義する手段を与えるものである．ここで定数は整数値，条件値，実数値，論理値あるいは文字列値となるような式であってもよい．また属性の列は当然ながら正しい組合せのものでなければならない．識別子と定数または識別子と属性の列との間の同値関係は，コンパイル時に確立される．コンパイル中にそのような識別子があると，その場所に同値関係を持つ定数あるいは属性の列があるかのように処理される．この文の構文は次の例から明らかであろう．

```
COPY('PARMS'); "Copy compilation parameters, including formal
               "definition for SERIES_1100
DEFINE SERIES_9000=NOT SERIES_1100;
    INCLUDE(SERIES_1100);
    DEFINE ATTRIBUTE CHAR_TYPE_SIZE=12 FIELDATA CHARACTERS;
    DEFINE TABLE_SIZE=32;
           ⋮
RESUME;
INCLUDE(SERIES_9000);
    DEFINE ATTRIBUTE CHAR_TYPE_SIZE=8 EBCDIC CHARACTERS;
    DEFINE TABLE_SIZE=16;
           ⋮
RESUME;
DECLARE SYSTEM_FILENAMES(TABLE_SIZE) : CHAR_TYPE_SIZE;
           ⋮
FOR I :=1 TO TABLE_SIZE DO
        BEGIN
           ⋮
        END;
```

変更が予測されるものは，定数や属性の列が現れる個所すべてをそのつど変更するよりも，その定数や属性の列を表す形式定義を行っておいて，その個所だけを変更するようにできる．さらに4.2節の例で示したように，1つのシステムから別のシステムに変換できるプログラムを，1つの原始プログラムだけで書くことも可能となる．PLUS言語で書いた移植可能なプロセッサの例としては，UTS 400 ターミナル・システムのための UTS 400 COBOL コンパイラがあげられる．ここでは DEFINE 文，INCLUDE と RESUME 文および他の PLUS 言語の要素と概念を用いて，UNIVAC シリーズ 1100, 90/700, 90/300 の間で移植可能なコンパイラの原始プログラムを一組だけですませている．

5.4 ENVIRONMENT 機能

UNIVAC シリーズ 1100 の PLUS コンパイラでは，ENVIRONMENT 機能を用いると，一度処理した宣言を保存しておいて，後の別のコンパイル操作時にそれを読み込んでくることができる．これを使えば，各モジュール間で共通の宣言があるとき，それを再処理する費用が節約できる．また変更する必要がない安定した宣言の集りをモジュールから取り除いて，ただ一度のコンパイルですますこともできる．モジュールをコンパイルするときには，前もって処理してある宣言をただ読み込みさえすればよく，コンパイル時間の節約になる．

```
MODULE;
        ⋮
CONTROL(ENVIRONMENT('ENVIR1')); "Declarations processed to this
                                "point will be saved in ENVIR1.
        ⋮
CONTROL(ENVIRONMENT('FILEA.ENVIR2')); "All declarations processed
                                      "to this point will be
                                      "saved in FILEA.ENVIR2.
        ⋮
TERM;

MODULE(ENVIRONMENT('FILEA.ENVIR2')); "Preprocessed declarations
                                     "from FILEA.ENVIR2 will be
                                     "read in.
DECLARE ...
TERM;
```

6. おわりに

PLUSは，データの抽象的な表現と機械に依存した表現とを調和させるために，さまざまな段階のデータ宣言を利用している．プログラムを抽象的な属性と構文だけで組み立てることは可能である．しかし，外部で定義されているデータの形式を導入する場合（たとえば ER 命令のパケット）には，正確なデータの形を指定する機能が必要となってくる．PLUSでは，このような必要に応えて，詳細な表現と機械的な表現ができるよう設計されている．詳細な表現は機械の性質を仮定してはいるが，このデータに対する演算は必ずしもその機械の性質に基づくわけではない．機械の性質に依存してデータを使用する場合には，MACHINE 属性によって明確に文書化しなければならない．宣言によって指定しない性質は，省略時解釈によって何も補われない．したがってプログラマは意味と使用法を考慮して，注意深くデータを宣言しなければならない．このことから，個々のデータの使用水準がプログラム上で明確に区別されるという効果がある．

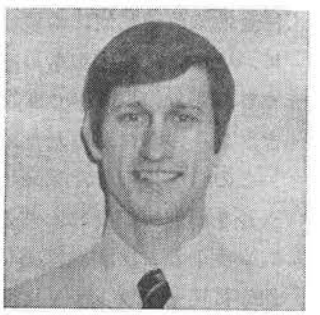
以上に加えて，インライン機能とか ASSIGNED 属性のような機械に特有の道具立てについても述べた．これらを PLUS に採用した目的は，高水準言語から標準的でないコードを生成したり，特定のレジスタを識別したり使用したりするためである．PLUS のコンパイル時の機能についてはすべてを述べたわけではないが，紹介したものだけでも十分その機能の豊富さが理解していただけよう．

なお，PLUS は，現在のところ社内のツールに限られ，顧客には提供していない．

(1100 ソフトウェア開発部　真田 正二 訳)

執筆者紹介　F. W. ストドーラ (Frank W. Stodola)

1969年 Minnesota 大学数学科卒業．1971年同大学コンピュータ科学の修士号を取得後，Sperry Univac 社に入社．以後8年間に同社 Roseville 開発センターにおいてランゲージ・システム・グループの一員として，MACRO, ASCII FORTRAN, APL, PLUS 等の言語プロセッサ類の設計とその実現に従事．現在，同グループの シリーズ 1100 PLUS, UTS 4 XX PLUS クロス・コンパイラ，および MCO V 77-800 FORTRAN の開発プロジェクトのプロジェクト・マネージャ．



ソフトウェア開発言語 RDL

RDL : A Language for Software Development

H. C. Heacox

要 約 近年，コンピュータ・コストの比率を見ると，ハードウェア・コストに比べソフトウェアコストが増大しつつある．この傾向はユーザの要求にも現れており，単なる機能の増強でなく，保守性の改善，使いやすさや安定性の向上，効率・費用・スケジュールの予測可能性というように，ますます高度なものを要求している．

これまで，ソフトウェアのライフ・サイクル全般にわたって適用できる包括的な標準，手続き，ツールがなかったため，求められた機能に十分応えられなかった．

この点を重視した Sperry Univac 社は，統一的かつ包括的で自動化されたソフトウェア開発ツールを研究・開発した．これらのツールのうちの1つが本稿の主題である．

ソフトウェア開発言語 (RDL : Requirements and Development Language) とは，要求仕様から保守に至るソフトウェア開発のすべての面が記述できるようにした非手続き言語である．RDL では統合データベースの構築のため，ステートメントを段階的に用いる．このデータベースには開発プロジェクトに関するすべてのデータが含まれる．したがって，データベースからいくつかの標準的報告書が作成できる．それは，要求／設計の仕様の分析，問題領域の追跡，文書の作成，最新状況の情報提供，全開発過程でのコミュニケーションとコーディネーションの確実化等である．RDL は，Michigan 大学の ISDOS プロジェクトの成果をもとに，Sperry Univac 社のソフトウェア開発部門が開発したもので，社内のツールとして使用されている．

本稿では，RDL で可能な言語構成を論じ，設計仕様の分野での詳細な例を示す．さらに，統合データベースの柔軟性や RDL プロセッサの報告書作成機能の能力を例を用いて説明する．

Abstract Industry trends reflect a continuing and significant shift in resource expenditures (funding, personnel, facilities, etc.) from hardware to software. In line with this trend, customers are becoming increasingly sophisticated in their demands not only for increased functionality, but also for improved maintainability, enhanced ease of use, greater stability, and predictability of schedules, cost, and performance.

Historically, the lack of comprehensive standards, procedures, and tools, applicable to the entire software life cycle, has been a major stumbling block in providing the desired features. Sperry Univac recognizes this need and is meeting it by creating a unified, comprehensive set of automated software development tools. One of these tools is the subject of this paper.

The Requirements and Development Language (RDL) is a non-procedural language which permits the user to describe all aspects of software development, from requirements specification through maintenance. RDL statements are used, incrementally, to build an integrated data base containing all data pertinent to the project. From this data base, standard reports can be generated to analyze requirements or design specification, to track problem areas, to produce documentation, to furnish up-to-date status information, and to ensure communication and coordination among all phases of the development process. RDL is based on work done by the ISDOS project at the University of Michigan. It was developed, and is being used as an internal tool for Sperry Univac Software Development.

This paper describes RDL by discussing the constructs available in the language, and by presenting detailed examples from the area of design specification. In addition, some examples are presented

Reprinted from ACM SIGPLAN NOTICES, (December 1979), Vol. 14, No. 12, pp. 71-79.

which illustrate the flexibility of the integrated data base and the power of the report-generating facilities of the RDL Processor.

1. はじめに

ソフトウェアの開発過程に対する支援系の例は数多くある．あるものは方法論で，それ自体重要ではあるが，支援するための自動化されたツールを持たない[1,2,3,8,9]．また，あるものは自動化されたツールを持っているが普遍性がない[4,5,6,7]．開発されるシステムが小さい場合それらでも間に合うが，システムの大きさや複雑さが増すにつれて次のような問題が生じる．

1) システムが大きく，複雑になれば，より多くの開発要員が必要となる．この場合，システムの設計と製作は別々のグループが担当することになる．またシステムのモジュール化により，製作グループがさらに増える．これらグループ間でのコミュニケーションやコーディネーションが欠けると，誤解を招いたり，大きな損失をもたらす結果となり，さらには信頼性のないソフトウェアがつくられたりする．
2) 標準の実施がますますむずかしくなる．とりわけ相互に関連性のない，たくさんのツールによって支援されるような開発過程では，標準を広範囲に適用することが非常にむずかしい．
3) 文書化は事後にツールごとになされるのがふつうなので，ドキュメンテーションを最新の状態で保ちにくい．
4) システム設計および製作の適切な情報を欠き，往々にして単体テストより上のテストでは計画性がなくなる．

以上の問題からも，開発過程自体の制御が困難なこと，ソフトウェアのバグが開発過程の後期に発見されること，そのデバッグに非常に費用がかかること，文書が貧弱でシステムを保守し拡張するのがむずかしいこと，などがわかる．

これらのことから，ソフトウェア開発についての総合的な方法論を用い，全過程に系統的に適用できる単一のツールの必要なことがわかる．ソフトウェア開発言語 RDL はそのようなツールである．

1.1 RDL の目的

RDL は非手続き言語であり，ソフトウェア開発のすべての面の記述ができる．それらには要求仕様，設計仕様，製作，検査の管理，計画，管理構造および職責等が含まれる．たとえば，プロジェクトの全情報を含むオンラインの総合データベースを除々に構築するのに RDL が用いられる．このデータベースは，プロジェクトが発足した当初から，関与している全グループの主な情報源となる．すなわち，このデータベースに対して問合せができ，そこから報告書が作成できる．また，自動的に文書を作成することができる．

1.2 RDL の歴史

Michigan 大学の ISDOS プロジェクトは開発記述言語 (PSL : Problem Statement Language)[7]と呼ばれる言語を開発した．しかし，RDL に要請されるものの方が，PSL の目的とする範囲より広いため，ISDOS ソフトウェアを利用できる新言語を設計するか，PSL を拡張するか，のどちらかが必要であった．幸いなことに，ISDOS システムは目標言語を超言語によって定義している．したがって，言語を定義する者が目標言語を利用者の要求に合わせられる．

Sperry Univac 社のソフトウェア開発部門は，PSL に基づいて，RDL を開発した．RDL

は社内のツールに限られ，現在のところ顧客には提供していない．

1.3 RDL の構造

RDL には，次の3つの基本構造がある．

1) “対象”とは，この言語の基本的構成要素である．対象の型は言語を定義するときに指定される．たとえば，DESIGN-FUNCTION はシステムの能動的要素(すなわち最終的にはコードとなるもの)を表す設計仕様での対象の型である．対象の具体値は利用者が一意で名づけ，RDL ステートメントによって *assign-scratch-file* 等と指定する．

2) “属性”とは，それが付随している対象を説明する記述である．属性の型は言語を定義するときに指定される．これに対して，属性の具体値は，利用者が値を与えて完成し，希望の対象に付与する．たとえば，IMPLEMENTATION-LANGUAGE 属性は，特定の DESIGN-FUNCTION 対象を製作するときに用いる言語を指定する．すなわち FORTRAN 等という値を持つであろう．また，属性は任意の長さの文章形式の値を持つように定義することもできる．

3) “関係”は2つ以上の対象を論理的関連によって結合する．したがって設計者は，

one DESIGN-FUNCTION INVOKES *another*

によって，他の DESIGN-FUNCTION を呼び出すことや，

a DESIGN-FUNCTION DERIVES *some data structure*

によって，ある DESIGN-FUNCTION があるデータ構造を導き出すことなどを指定できる．

対象・属性・関係のそれぞれの型はソフトウェア開発のすべての面を覆うように定義されている．しかし，RDL は比較的新しい言語なので，しばらくは言語定義の過程が続くであろう．

2. RDL の使用法

ソフトウェア設計段階での例を用いて，RDL の使用法を説明する．ただし，例の中では次の対象を用いる．

1) DESIGN-FUNCTION 対象：これは1.3節で説明したとおりである．
2) DATA 対象：システムのデータ構造を表す．
3) REQUIREMENTS-FUNCTION 対象：これは DESIGN-FUNCTION 対象に

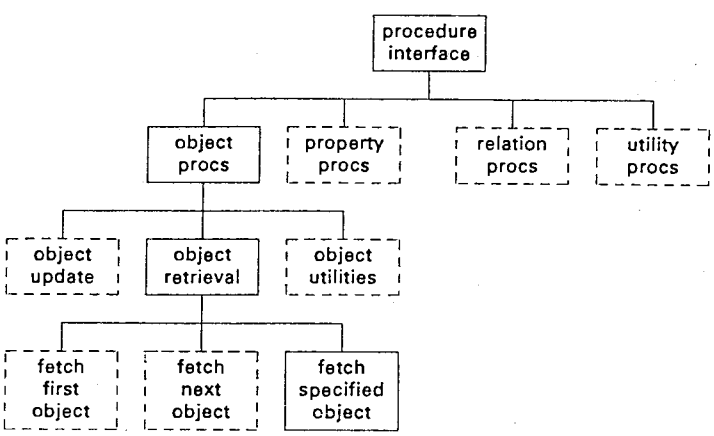


図1 手続き呼出しインタフェース
Fig. 1 The RDP procedure interface

似ているが，要求仕様において述べられた機能を表す点が異なる．

4) ORGANIZATION 対象：開発組織の管理構造を記述する．

例の中では，RDL のキーワードは大文字で示し，利用者の定義する構成要素(対象の具体値の名前等)は小文字のイタリック体で示す．感嘆符(!)は命令文の終止符として用いている．

2.1 高水準設計仕様

ソフトウェア開発における有効な概念の1つに下降型設計がある．RDL がこの技法に役立つことを以下に示す．いま RDL データベースへの手続き呼出しをするインタフェースの設計作業を仮定する．この構造を下降型技法に従うと図1のように表せる．

最高水準の対象(手続きのインタフェース自体)の RDL による記述は次のようになる．

DEFINE DESIGN-FUNCTION *procedure-interface*!
DESIGN-LEVEL IS SUBSYSTEM!
TITLE!
RDL Procedure Interface!
PURPOSE!
The procedure interface provides access to the RDDB via FORTRAN subroutine calls. This allows the user to write applications against the data base, either for report generation or for making data base updates from within his applications.!
RESPONSIBILITY OF *RDL-Group*!
KEYWORDS ARE *proc-int*!
PARTS ARE *object-procs, property-procs, relation-prcos, utility-procs*!

DEFINE 文は，もし *procedure-interface* という名の DESIGN-FUNCTION 型の対象がまだデータベース中に存在しなければ，それを生成させる．この命令文は文脈を確立する役割も果たす．すなわち，この命令文以降，次の DEFINE 文までのすべての命令文がこの *procedure-interface* DESIGN FUNCTION に適用される．

DESIGN-FUNCTION の属性の生成は DESIGN-LEVEL, TITLE, PURPOSE の各文によって行われる．DESIGN-LEVEL は，属性の一例であり，その値は31文字以内で RDL の定義で許された値の表にあるものでなければならない．一方，TITLE および PURPOSE は，属性の値として文章の長さが自由で，データベース中に占める領域だけを考慮すればよい属性の例である．これらの属性により，対象のさまざまな面を人が読みやすい形に自由に表すことができる．このことは文書の自動作成に非常に役立っている．

残りのステートメントは RDL の対象間の関係の仕様を説明している．PARTS ARE 関係によって設計階層を指定でき，より下位へ，より細部へ，と設計の詳細化を行ったことを反映できる．PARTS 関係は言語定義において，同じ対象型間の $1:n$ 関係として指定されている．したがって，これは階層構造をつくりだす．関係は $m:n$ と定義することもでき，網構造をつくることも可能である．

RESPONSIBILITY 関係は DESIGN-FUNCTION 対象の具体値を ORGANIZATION 対象の具体値に結びつける．これによって管理に適した制御構造が定義される．同様に KEYWORD 関係は KEYWORD 対象の具体値を他の対象と結びつけ，データを論理的に区分する．データベースから報告書を作成するとき，この関連が使われる．

2.2 設計仕様の詳細化

2.1節に指適したように，最高水準の設計の詳細化は，PARTS 関係によって追跡できる．*procedure-interface* 対象は，その部分の1つに *object-procs* と呼ばれる DESIGN-FUNCTION を持っている．それは，また *object-retrieval* という下位部分を持ってい

る．この詳細化は次の例により示される．

```
DEFINE DESIGN-FUNCTION object-procs!
DESIGN-LEVEL IS PACKAGE!
TITLE!
Procedures to Manipulate Objects!
PURPOSE!
This package provides user-callable routines which allow the user to create, delete,
modify, and retrieve data-base objects!
RESPONSIBILITY OF RDL-Group!
KEYWORDS ARE proc-int, proc-int-objects!
PARTS ARE object-update, object-retrieval, object-utilities!

DEFINE DESIGN-FUNCTION object-retrieval!
DESIGN-LEVEL IS PACKAGE!
TITLE!
Object Retrieval Package!
PURPOSE!
This package provides routines which allow the user to:
  1. Retrieve the first object in the data-base.
  2. Retrieve the next object in the data-base.
  3. Retrieve a specified object by name.!
KEYWORDS proc-int, proc-int-objects!
RESPONSIBILITY OF RDL-Group!
PARTS ARE fetch-first-object, fetch-next-object, fetch-specified-object!
```

この例の中で定義されている2つの対象はともに，PACKAGE という値の DESIGN-LEVEL 属性を持っている．また，それらの位置は第2，第3水準の設計対象であることを表している．

2.3 コード水準の設計

設計の詳細化の水準を順次降ろしていくと，設計対象をコード化できるようになる．そのような対象は次の例で示される．

```
DEFINE DESIGN-FUNCTION fetch-specified-object!
DESIGN-LEVEL IS UNIT!
TITLE!
Fetch Specified Object!
PURPOSE!
Given the name of a user-defined object, fetch-specified-object returns the type
code of the object.!
ALGORITHM!
    flow            fetch-specified-object
    call            Call find-name to verify object's existence and get its type
    if              Error in find-name?
        process     Zero type field in packet
        process     Set error status
    else
        process     Set type field in packet
        process     Status=0
    endif
    endflow!
PROVIDES specified-object-retrieval!
RESPONSIBILITY OF RDL-Group!
KEYWORDS ARE proc-int, proc-int-objects!
FORMAL-PARAMETERS ARE object-packet (UPDATE), status (OUTPUT)!
INVOKES find-name PASSING object-packet (INPUT), object-type (OUTPUT)!
UPDATES object-packet USING object-type, object-status!
```

この例で，DESIGN-FUNCTION *fetch-specified-object* は UNIT の DESIGN-LEVEL であり，コーディングに適していることを示している．この水準には新しい言語構造がいくつか現れている．1つは ALGORITHM 属性で，その値は DESIGN-FUNCTION の算法を記述する文章である．上の例ではブロック構造の擬似コードを用いている．このようにして，人が読める形でそのコードのオペレーションをデータベース中に文書化することができる．

次に，この例に現れる新しい“関係”を考える．FORMAL-PARAMETERS 関係は DESIGN-FUNCTION の呼出し系列を記録するために用いられる．これは重要な配慮である．というのは，それが FORTRAN サブルーチンとして製作されることになっているからである．INVOKES 関係は，システム中にあるさまざまなサブルーチン間の CALL の関連を指定するとともに，呼出しの際に渡される実引き数を記録する．INPUT, OUTPUT, UPDATES という値は呼出しの境界を越えて流れる情報の方向を記述する．

UPDATES 関係は，DESIGN-FUNCTION 対象によって行われる DATA 対象の処理を，設計者が記述できるようにしている．UPDATES の他に，この言語(RDL)で利用できる関係に DERIVES や DELETES 等がある．この例の中の UPDATES 関係には，値が変更させられる DATA 対象とともに，更新の計算中に使われる DATA 対象も指定されている．

PROVIDES 関係によって要求と設計の間の境界をまたがった伝達ができる．おそらく，これは要求仕様の作成過程のある時点でデータベースから指定した対象を取り出す能力が必要であると認められたのであろう．この要求は *specified-object-retrieval* という名の REQUIREMENTS-FUNCTION 対象を生成することにより指定されている．PROVIDES 関係は，この REQUIREMENTS-FUNCTION と，それを実現する DESIGN-FUNCTION との間の結びつきを形式化している．

3. データベースの使用

RDL データベースの使用に際し，次の3つの重要な概念が適用されている．

1) 各対象の唯一の具体値はデータベース内にある．したがって，対象間の関係が生成されるときはいつでも，関与するすべての対象がはっきりと結びつけられ，それらのどこからでも関係の具体値を検索し始めることができる．
2) データベースは情報が利用可能になるに従って，徐々に構築することができる．また条件が変わったときには容易に変更できる．
3) データベースを系統的に分析し違反するものがあれば，その報告書を自動作成する．

これらの各々の概念は以降の節で議論される．

3.1 関係の補完形式

対象の具体値がただ1つしかないので，RDL の対象の間に関連が確立されたときには，それに関与しているどの対象からもその関連が見えるようになる．関連の解釈は，主な関心がどの対象にあるかによって異なる．たとえば，2.1節で示した *procedure-interface* 対象に対する PARTS ARE 文を考えてみよう．この場合 *procedure-interface* は，下位部分に *object-procs* 等を持つ上位部分である．*object-procs* から見ると，この関係はあたかも次の命令文が登録されているかのように考えられる．

DEFINE DESIGN-FUNCTION *object-procs*!
PART OF *procedure-interface*!

procedure-interface と *object-procs* を結合している PARTS 関係の具体値はただ1つである．しかし，この関係は2つの異なる見方に対応している．

この概念の有効性は，いったん関係が確立されるとそれに関与したすべての対象からその関連が見える，という事実にある．たとえば，ある設計者が

some DESIGN-FUNCTION INVOKES *another*

と指定したとしよう．そして，呼び出される DESIGN-FUNCTION はだれか他の設計者の責任であったとしよう．呼び出される対象に責任を持つ設計者がその関係の生成されたことをたとえ知らなくても，上記の関係が存在することは呼び出される対象からも見える．その情報はデータベース中に存在するのだから，呼び出される対象の設計者もそれを知ることができる．これは 3.3 節の最初の例にも示されている．

3.2 データベースの段階的な指定

RDL データベースには，作業を開始するために利用者がデータベースに入れるべきデータを完全に指定する必要がない，という重要な機能がある．情報は利用可能になったときにどのような順序で入れてもよく，増やしたり，削ったり，変更したりすることがいつでも可能である．

たとえば初めの段階で，実現されるはずの *catalog-file* という DESIGN-FUNCTION は *file-directory* という値の DATA 対象を必ず更新することがわかっているとすれば，次の命令文がデータベースに入れられる．

DEFINE DESIGN-FUNCTION *catalog-file*!
UPDATES *file-directory*!

後になって，更新の計算は *file-name* という DATA 対象の値によることがわかり，*catalog-file* は *status-return* という DATA 対象を導き出すことがわかったとしよう．この情報は次の命令文によって追加することができる．

DEFINE DESIGN-FUNCTION *catalog-file*!
UPDATES *file-directory* USING *file-name*!
DERIVES *status-return*!

これらの命令文を処理している間に，次の2つの処置がとられる．

1) *catalog-file* と *file-directory* との間に現存する UPDATES 関係は修飾をする関連 “USING *file-name*” を追加することによって変更される．
2) *catalog-file* と *status-return* の間に新たな DERIVES 関係が追加される．

同様にして，属性の具体値を変更したり，新しい具体値を登録したりできる．

3.3 データベースの分析と報告

これまでに述べてきたような形式言語を用いることによって，要求や設計等の表現に一定の紀律を課すことができる．それ以外にも RDL には，ソフトウェア・プロジェクトに関するすべての情報がデータベースに蓄えられる，という重要な利点がある．そのデータベースからは最新の情報がたやすく取り出される．またそのデータベースのためにつくられた応用プログラムによって，分析および報告書作成が系統的かつ自動的になされる．これらは，単純な問合せから影響分析や一貫性 (consistency) 検査，完全な文書の作成までの範囲の応用ができる．標準的な分析と報告のためのプログラムが多数つくられており，それらは一括して RDL プロセッサあるいは RDP と呼ばれている．

次の例を考えてみよう．

1) *catalog-file* という DESIGN-FUNCTION を修正していて，その変更の影響が分析できるよう，どの DESIGN-FUNCTION がそれを呼び出すのか知りたいとす

る．これは，標準の RDL プロセッサに次のコマンドを与えればできる．

```
LIST NAMES SELECTION='INVOKES catalog-file'
```

この結果，指定された対象を呼び出すすべての DESIGN-FUNCTIONS の名前の表がつくられる．

2) 「DESIGN-LEVEL が PACKAGE であるような DESIGN-FUNCTION は他の DESIGN-FUNCTIONS を呼び出してはならない」と利用者が一貫性の部分で定義したとしよう．次のコマンドを用いれば，一貫性の報告書が得られる．

```
LIST NAMES SELECTION='DESIGN-LEVEL
                      =PACKAGE AND INVOKES'
```

結果として，PACKAGE レベルの対象('DESIGN-LEVEL-PACKAGE' で指定されたもの)のうち INVOKES 関係の中に呼出し側として加わっているもの，すなわち一貫性の定義に違反する対象すべての名前の表が得られる．

3) 関係によって課せられた構造は，階層構造であっても，網構造であっても，次のコマンドによって得られる．

```
LIST STRUCTURE NAME=procedure-interface
               RELATION=PARTS-ARE
```

データベースが第2章の言語の例によって生成されたものとすると，このコマンドの結果は以下に示された構造報告書の例のようになる．その構造は図1のダイヤグラムと同じであることに注意されたい．

```
                STRUCTURE REPORT FOR OBJECT procedure-interface
                  RELATION PARTS-ARE JAN 10, 1979

procedure-interface PARTS-ARE
    01  object-procs
        02  object-update
        02  object-retrieval
            03  piffob
            03  pifnob
            03  pifsob
        02  object-utilities
    01  property-procs
    01  relation-procs
    01  utility-procs
```

4) いくつかの対象に関するすべてのデータが RDL 文の形で表示されている，単純ではあるが効果的な報告書は簡単につくれる．このような報告書は，対象に対して明確に指定し登録されたデータのみならず，他の対象を記述するために登録された関係，指定された対象を含むものがあれば，その補完形をも表示する．たとえば，次のコマンドを考えてみよう．

```
LIST RDL NAME=object-procs
```

このコマンドは以下に示される出力をつくりだす．たとえ，"PART OF *procedure-interface*" という命令文が *object-procs* の記述に際して登録されなかったとしても(2.2節参照)，2.1節の例における "PARTS ARE *object-procs*" 命令の補完形として生成されているので，以下に示す LIST RDL に対する報告書の例の中にはそれが現れていることに注意されたい．

```
DEFINE DESIGN-FUNCTION object-procs!
DESIGN-LEVEL PACKAGE!
TITLE!
Procedures to Manipulate Objects!
```

PURPOSE!
This package provides user-callable routines which allow the user to create, delete, modify, and retrieve data-base objects!
RESPONSIBILITY OF *RDL-Group*!
KEYWORDS ARE *proc-int, proc-int-objects*!
PARTS ARE *object-update, object-retrieval, object-utilities*!
PART OF *procedure-interface*!

この他に種々の文書を作成する報告書プログラムがいくつかある．説明は紙面の都合で省略するが，主な特徴を次に示しておく．

・報告書プログラムは文書を自動的に生成することができる．したがって，単に費用や時間がかかりすぎる等の理由から新しい版がつくられず，文書が陳腐化するということがない．

・報告書プログラムは，合成した関係や単一の関係のレベルを広範囲に探索して得られたデータを要約することができる．たとえば，データ構造の設計の記述において，DATA 対象間の関連を通して階層構造や網構造が指定できる．報告書プログラムの構造を簡潔に示すと表1のようになる．

表 1　データ構造の例
Table 1　Sample data structure

名　称	形　式	長さ
property-entry	表	
common-entry	表	
next-pointer	整　数	2バイト
keyword	文　字	31バイト
form-pointer	ポインタ	4バイト
relation-entry	表	
common-entry	表	
next-pointer	整　数	2バイト
keyword	文　字	31バイト
form-pointer	ポインタ	4バイト
relation-type	整　数	1バイト
nbr-of-parts	整　数	1バイト

・報告書プログラムによって作成された出力は，PERT プログラム，コンパイラ，連結編集プログラム等のような，他のプロセッサの入力用に形式を自動的に整えることができる．

4. おわりに

RDL は非手続き型で，機械処理可能な言語であり，ソフトウェア開発の全フェーズを記述することを意図している．このような言語およびそれから導き出される統合データベースは，明らかに開発過程の管理において大きな可能性を持っている．Sperry Univac 社内の使用経験によって，このシステムが

1)　管理面からの開発過程の可視性と制御の向上
2)　生産性の向上
3)　ソフトウェアの品質向上

の点で大変有効であることが示された．

本稿の作成に当たり，ご協力いただいた Sperry Univac 社の W. L. Caudle, M. Feuer,

R. M. Moore, R. D. Vavra の各氏に感謝する.

（日本語システム開発部　荻原 智弘 訳）

参考文献

[1] F. T. Baker, "Chief Programmer Team Management of Production Programming", *IBM Syst. J.*, Vol. 11 No. 1, 1972, pp. 56-73.

[2] V. Basili and A. J. Turner, "Iterative Enhancement : A Practical Technique for Software Development," *IEEE Trans. Softw. Eng.*, Vol. 1, No. 4, Dec. 1975, pp. 390-396.

[3] B. Boehm, "Seven Basic Principles of Software Engineering," *Infotech State of the Art Report on Software Engineering Techniques, 1977*, Infotech International Ltd., Maidenhead, UK, 1976.

[4] S. H. Caine and E. K. Gordon, "PDL——A Tool for Software Design," *AFIPS Conf. Proc.* Vol. 44, 1975, NCC, pp. 271-276.

[5] C. G. Davis and C. R. Vick, "The Software Development System," *IEEE Trans. Softw. Eng.*, Vol. 3, No. 1, Jan. 1977, pp. 69-84.

[6] T. A. Dolotta and J. R. Mashey, "An Introduction to the Programmer's Workbench," *2nd Int'l Conf. Software Engineering*, 1976, pp. 164-168.

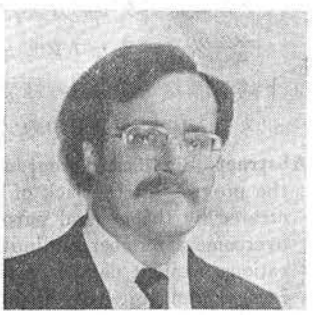
[7] D. Teichroew and E. A. Hershey, "PSL/PSA : A Computer Aided Technique for Structured Documentation and Analysis of Information Processing Systems," *IEEE Trans. Softw. Eng.*, Vol. 3, No. 1, 1977, pp. 41-48.

[8] N. Wirth, "Program Development by Stepwise Refinement," *Comm. ACM*, Vol. 14, No. 4, Apr. 1971, pp. 221-227.

[9] N. Wirth, "On the Composition of Well-structured Programs," *Comput. Surv.*, Vol. 6, No. 4, Dec. 1974, pp. 247-259.

執筆者紹介　H. C. ヒーコック (Harry C. Heacox)

Carnegie-Mellon 大学を卒業後, Wisconsin 大学でコンピュータ・サイエンス分野の Ph. D を取得. Sperry Research Center でデータベース管理とソフトウェア工学に従事. その後 Sperry Univac 社へ移り, ソフトウェア開発技術部門でプロフェッショナル・コンサルタントとして活躍, RDL 言語や RDL プロセッサの設計および開発を担当.



データベース・システムの最近の傾向

Current Trends in Data Base Systems

G. A. Champine

要 約 以下の主要な4分野におけるデータベース・システムに関する最近の技術動向について述べる.

大容量記憶域：この分野では記録密度の改善が著しく，この傾向は継続するであろう．また，適用される技術は薄膜ヘッド，純金属媒体の採用を含み，ビット当たり費用は記録密度に逆比例して低下するであろう.

利用者インタフェース：ネットワーク，ハイラキカル，リレーショナルの3つのデータモデルが支配的であるが，共通のデータベース構築に関する最近の研究によって，モデル間の一対一変換が可能であり，各データモデルを単一の記憶構造によって支援できることがわかった.

分散データベース：応答時間と可用性に改善が見られる一方，通信費用が削減されるようになる．各ノードへのデータの配置法には2つの主要な方法(複製データ配置と分割データ配置)がある．複製データ配置は高い信頼度を示すが，更新の頻度が高いと性能が劣化する．一方，分割データ配置はアクセスが非常に局所的ならば有効である.

データベース・コンピュータ：これは従来のコンピュータに対して，その専用の構築と連想アドレスによって4つの性能上の改良が実現できて，ホストに対するバックエンド，あるいは分散システムにおける1つのノードとして使用可能である.

Abstract Significant progress is being made in four major areas of data base systems. The reason for the progress in the lack of substantive barriers to continued technical improvement, and a large market created by the general purpose nature of the systems. Data base systems were developed initially to overcome a number of limitations in file systems by providing storage design independent of applications, explicit data definition, shielding of data formats and storage structures from the users, and recovery and integrity independent of applications.

1) In the mass storage area, the recording density of moving head disks has historically doubled every 30 months for the last 10 years, and will continue to improve at a rate approach to the historical trends. The enabling technology includes thin film heads, pure metal media, and smaller physical tolerances. The cost per stored bit should continue to decline in inverse proportion to the recording density.
2) 3 data models dominate the user interface=these are the network, hierarchical, and relational. Recent work on the common data base architecture has shown that a one-to-one correspondence exists between features in the network data model and the relational data model, allowing translation from one data model to another and the support of both data models by a single storage structure.
3) Distributed data bases offer improvements in response time and availability while reducing communications cost. The two principal ways of allocating data to nodes are ① replicated, and ② partitioned data allocation. Replicated data has better reliability but may sacrifice performance in update-intensive applications. Partitioned date depends on high locality of access for efficient operation.
4) Data base computers can provide a factor of four inprovements in performance due to specialized architecture and associative addressing relative to conventional machines, and are used as back ends to a host or as a node in a distributed system.

1. はじめに

現在，データベース・システムには，急速に利用者が増えていることと，論理的・物理的技術や構築段階の技術が急速に進歩していることの2つの著しい傾向がある．これらは，低価格の直接アクセス大記憶域が利用可能になったことや，情報の蓄積と検索に効果的なシステム構築への関心がかなり高まっていることと深い関係がある．

データベース・システムは既存のファイル管理システムにおける多数の欠陥を克服するために開発されてきたものであるが，そのアプローチの方法にはいくつかある．

そのうち，リレーショナル・モデル，ハイラキカル・モデル，CODASYL のデータ設計言語に見られるネットワーク・モデルの3つが主なものとなっている．

本稿では，データベース・システムに関する4つのカテゴリ，すなわち大容量記憶域としてのハードウェア，利用者のインタフェース，分散データベース・システム，データベース・コンピュータにおける技術進歩について考える．

2. 大容量記憶域としてのハードウェア

現在の記憶システムは，乱アクセス（または主）記憶域，直接アクセス記憶域，順アクセス記憶域という階層からできている．

2.1 主記憶域

主記憶域には直接アクセスまたは順アクセス記憶域の操作に要するバッファ域がとられる．今日の主記憶域は，10年前の大容量記憶域の容量と同程度の大きさを持つようになっている．主記憶域が安価になるにつれ，最適の直接アクセス記憶域と順アクセス記憶域の割合はより大きな主記憶域で均衡するようになった．その結果，プロセッサの利用度は急速に増大しており，今日では85～95パーセントが普通である．また主記憶域のサイズの増大に伴い，より多くのデータが主記憶域に蓄えられ，記憶域階層の低段へ移送されなくなったため，入出力の実行度も減少している．記憶域に使用するチップの改良は集積度の向上と費用の低下をもたらし，この傾向は今後も続くであろう．

最近の写真食刻技術の改良によって，262キロビットのチップが使用可能となり，また1メガビットのチップも将来実現可能である．この費用低減によってあらゆる種類のコンピュータ・システムの主記憶域はさらに大きくなるであろう．

2.2 直接アクセス記憶域

伝統的に固定ヘッドまたは移動ヘッドの磁気ディスクが使われたが，最近，2つの新しい固体直接アクセス技術が現れた．すなわち，磁気バブル素子とレーザを使った光学記憶装置である．後者はいくつかのプロトタイプが開発された．この技術によって近い将来に製品化の可能性もある．

磁気ディスクの容量は，記録密度の増大に伴い急速に増加している．記録密度はディスク上の磁性被膜，ヘッドと記憶面の間隔，ヘッドの磁極間隙によって決まる．これらは新技術によってさらに縮められ，記録密度は増大するだろう．ディスクの磁性被膜の分解能を高くするためには，粒子特性の制御を改良する方法か，表面に金属メッキ等を行って品質を高める方法が用いられる．A. S. Hoagland はこのトピックスを詳しく論じている[13]．

読出し・書込みヘッドの技術も，とくに薄膜ヘッドを中心にかなり注目されている．薄膜ヘッドによって低費用と高性能の双方が実現できる．薄膜ヘッドは一括生産により費用が低減でき，仕様諸元にわたって精密な制御が実現可能で高性能となる．これらの技術の進展によって，30か月ごとにビット当たり費用が半減するという傾向が，少なくとも数年

は続くと考えられる．一方，1スピンドル当たりのディスク容量は1982年までには1ギガバイトを越えるものと考えられる．

2.3 順アクセス記憶域

磁気バブル素子は，順アクセス技術であり，密度を急速に増加させる．1チップ当たり1メガビットのバブル装置が発表されたが，1990年までには16メガビットのチップができると考えられる．しかし，バブル素子のチップが高密度になるに従い，そのアクセス速度は遅くなる傾向があり，現在は，250キロビットのもので，7ミリ秒程度である．磁気バブル素子は多分小型システムに使用することになろう．最近，Bell研究所ではバブル駆動用の回転磁界発生コイルを装置から取り除く改良を行った．この開発で速度は5倍に改善された．

3. 利用者インタフェース

利用者インターフェースとはデータベース・システムへの論理インタフェースのことである．この分野の研究では，次の点が目立っている．

1) データモデル
2) アクセス法
3) データの複数視野(ビュー)，スキーマ間変換，スキーマ，サブスキーマ，記憶域スキーマの分離等を支援する共通データベース構築

3.1 データモデル

現在，リレーショナル・データモデルとネットワーク・データモデルが主要なモデルである．ネットワーク・モデルは1970年頃に展開され，比較的に古く，CODASYL DBTGの仕様に代表される．このモデルでは利用者が，かなり詳しく記憶構造やアクセス経路やデータ構造を指定する必要があり，また，あらかじめ定義されていないアクセス経路は使用できないという意味で，融通性に欠ける．さらに，記憶構造は一般にリンク付きリストへのポインタで構成され，非常に複雑である．それに対して，リレーショナル・モデルでは記憶構造やアクセス経路は利用者から遮蔽され，アクセス経路もあらかじめ定義しておく必要がない．しかし，定義したアクセス経路がないと，検索性能が悪くなるおそれはまぬがれない．記憶域とデータの構造は単純であって，単一レベルのレコード列として扱われる．さらに詳しくは M. M. Astrahan らが論じている[2]．

約10年にわたって，ネットワーク・モデルは主流となっていたが，現在，リレーショナル・モデルへの関心が高まっている．このモデルの魅力は，単純だが強力なインタフェースを持つ点，あらかじめ定義されたアクセス経路以外による問合せが可能な点にある．現在リレーショナル・モデルの効率は低いが，ハードウェア費用が低下して現実的価値が高まれば，その魅力は増大する．さらに，以下に述べる連想記憶装置はリレーショナル・システムの性能を著しく改良する．

ハイラキカル・モデルは，レコードあるいは構造の集合を単一の論理路で検索できるように束ねた，ネットワーク・モデルの部分モデルである．

3.2 共通のデータベース構築

現在データベースに対するいろいろなアプローチの統一化と単純化の動きが進行している．この動きは過去数年にわたる計算機言語で行われる統一化と単純化に似ている．この方向での1つの仕事は共通データベース構築である．この構築は，今日使われているデータベース・システムの理解と記述のための統一的基礎を与えるほかに，以下に述べること

を可能とする.

1) ネットワークとリレーショナル双方のシステムの各特性が互いに同値であることの証明
2) データモデル間のデータ定義と操作命令の変換
3) データモデル間のスキーマの変換
4) 一方のデータモデルで構成したデータの記述と操作を他方のモデルで構成したスキーマ上へ重ね合わせること
5) 1つのシステム内で唯一のデータ構造を支援すること
6) データ操作命令を異種の分散システムの間で変換すること

共通データベース構築の方法論は，異なるデータモデル間（あるいは同一データモデル内の異なるレベル間）に存在する共通性を明らかにすることができるであろう.

3.3 複数のデータモデルの支援

von Neumann 型計算機における現在のデータベースの実現において，ネットワーク・データモデルはポインタによって実現され，リレーショナル・データモデルは表探索によって実現される．ポインタをたどるアクセスについては，良好な構築設計の研究がほとんど進んでいない．一方，表の速い探索法については，かなり研究が進んでいる．レコード間の結合は，ネットワーク・モデルではポインタにおいて行われ，リレーショナル・モデルでは属性値を表す1つのフィールド（共通キー）が同一の値を持つことにより結合される．このポインタと共通キーの値とは同一の情報を含んでおり，相互に交換して使用する解釈が可能である．たとえば，リレーショナル・モデルをポインタで構成した場合，レコードの共通フィールド値を結合するのにそのポインタが使用でき，フィールド値の集合を連結して，フィールド値が集合の開始点でただ1回現れるようにできる．しかし，フィルード値を集合のすべての要素レコードへ複写してポインタを除くと，ネットワーク・モデルをポインタでなく共通フィールド値を使って実現することもできる．すなわち，どちらの記憶構造（ポインタまたは共通フィールド値）でも両方のデータモデルが実現可能となる.

3.4 単一データベース構築

共通データベース構築のもう1つの結果は，システム内では単一の構造をデータに適用して，ファイルやデータベースを分離しない点である．そこで複数のファイル操作言語が1つのデータベースによって同等に支援できることになる．たとえば，ネットワーク・モデルを使うとすると，ファイル宣言はデータベース・サブスキーマ，スキーマ，記憶スキーマと生成し，読込みと書出しはデータ操作命令を生成する．この操作命令は正規のデータ操作言語のコンパイラでコンパイルされ，データとのインタフェースで変更を必要としない．一方，データはデータベース自体のインタフェースを通してもアクセス可能である.

共通データ構築の適用によって，最終利用者によく適合し，実現も理解も単純なデータモデルができあがるように思われる．得られるモデルはおのおのの既存モデルの特性を包含するから，これらのモデルよりもさらに機能上強力になるに違いない．また共通データベースの構造とデータベース・コンピュータの間には密接な関係が存在する．経済的な実現可能性のためには，データベース・コンピューターはいくつかの異なるデータモデル，とくに，リレーショナルとネットワーク・モデルを含めて支援可能でなければならない．したがって，共通データベースの構造はこれを実現する基礎的なものとなる.

4. 分散データベース・システム

広範に分散したノードによって構成された，典型的な分散システムをさらに考察することにする．

4.1 目的と要求

分散システムのノード上にデータを分散する目的は，一般的には，分散システム自体が持つ全体の目的と同じであるが，次の目的はとくにデータベースの分散に関係する．

1) 性能：利用者の問合せに対する応答の速度
2) 低費用：データ通信の利用度の低下に伴う費用の低下
3) 分配可能性 (shareability)：地理的に分散したノード間でデータの分配が可能なこと
4) 信頼性：1点以上のノードを失ってもシステムが十分機能するように保証すること
5) アクセス透過性 (access transparency)：すべてのノードに一様な論理アクセスを行うこと
6) 適合性 (tunability)：もっともデータを使うノードへデータを分散記憶させること
7) 拡張可能性：既存ノードあるいは増設ノードによってデータベース規模を増大しうること

これらの目的のいくつか，とくに性能・信頼性・適合性・拡張可能性は冗長度の導入によって得られるが，冗長度は以下に述べるように完全性 (integrity) の問題を発生させる．また，以下の目的はほとんどの集中システムでも実現しうるが，これらは任意のトータル・システムにおける要求であり，分散システムでも実現されなければならない．

1) 回復 (recovery)：システムは，誤り，メッセージの損失，あるいは修復不能の故障から自動的に回復できること
2) 機密保護 (security)：システム管理者が強制するデータ保護のためのアクセス権を利用者が指定できること
3) 再編成：局所的または全域的に効率を改善するために，データを再編成できること
4) 効率：最大レベルの同時並行性を保証し，トランザクション破棄を低率にするために適切な算法を選択して，システム資源を効果的に使用すること
5) 首尾一貫性 (coherency)：データの多重コピーを矛盾なく維持すること
6) デッドロック：デッドロックによってトランザクション処理の完了を妨げないようにすること
7) 公正 (fairness)：全ノードは同等の優先度で維持すること

これらの要求は明らかにデータの分散によって困難で複雑になる．分散処理の環境でこれらの要求を満たす技法について述べよう．これらの要求は優先順位の高い順になっていないが，おそらく回復の要求がもっとも重要であろう．回復は現在の集中データ管理システムの基本的設計条件をなすが，分散システムではさらに重要性を増すであろう．他の要求もまた重要であって，どのようなプロトコルでもそれらをすべてを満たさなければならない．

4.2 集中対分散データベース

図1に示すように，伝統的な集中データベース・システムは次の要素からできている．ここでは，利用者の応用プログラムに含まれているファイル・システムも，特例としてデータ定義やアクセス法に含まれると考える．

1) 応用ソフトウェア
2) オペレーティング・システム

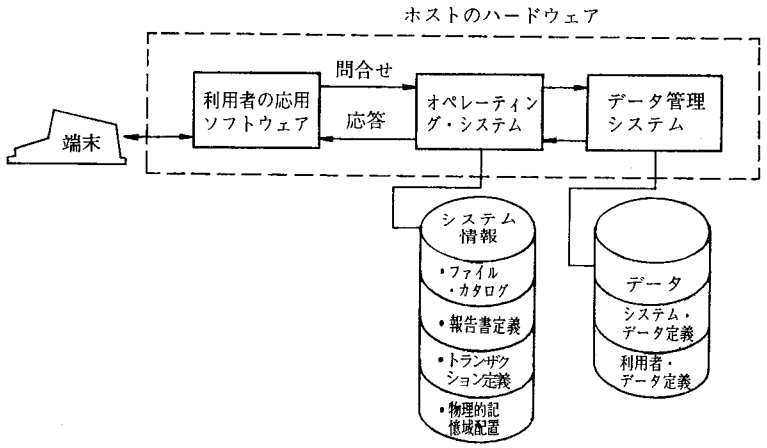


図 1　伝統的集中データベース・システムの構成
Fig. 1　Traditional centralized data base system organization

3)　データ
4)　物理的記憶域配置
5)　データ管理ソフトウェア
6)　システムのデータ定義
7)　利用者のデータ定義

ここで利用者とその応用プログラムは，オペレーティング・システムを介してデータ管理システムとインタフェースをとる．データ管理システムは全体の大容量記憶域を制御する．図2に示す分散データベース・システムは集中データ・システムと同一の機能要素を含むが，さらに，通信ポート，回線網データ・ディレクトリおよび回線網データ管理システムを含んでいる．一般に，通信ポートはノードの物理的位置と必要な経路指定の情報を持っており，その役割は分散システムでも集中システムでもまったく同じである．回線網データ・ディレクトリは，各ノードとそのノードにあるデータの関係を示す論理情報を含んでいる．また，回路網データ管理システムはデータの地理的分布の様子をすべて管理する．

これらの管理タスクには次の機能が含まれる．

1)　要求が局所的データに対するものか否かを決定するために，要求を傍受すること

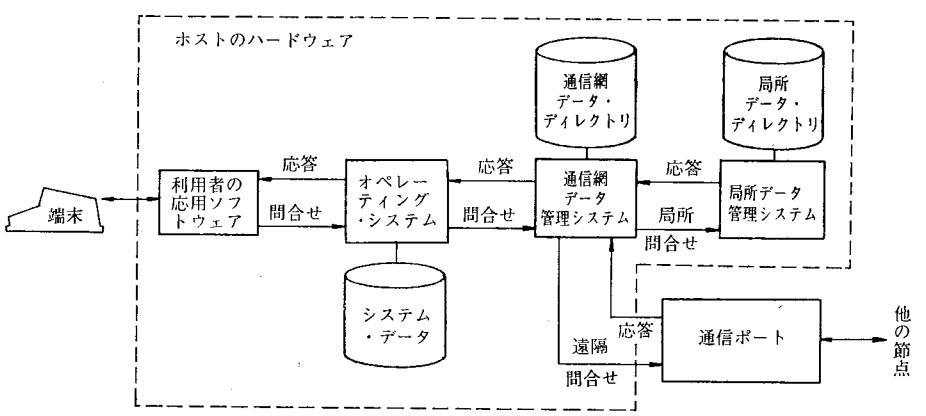


図 2　分散データベース・システムにおける節点
Fig. 2　A node within a distributed data base system

2) 回線網データ・ディレクトリをアクセスして，局所的でないデータについてはそれを保持するノードへ要求を回送すること

3) 1つのアクセスに複数のノードが関係するとき，すべての処理と応答を調整すること

4) 利用者や，局所的でかつ遠隔のデータ管理システム，あるいはシステム・ディレクトリとインタフェースすること

5) 異種の分散システム間での命令とデータの変換を行うこと

4.3 データ分散の方法

システム構造内にデータを分散するには，分割と複製の2つの基本的方法がある．データの分割とは，1つのデータの部分集合が，それぞれのノードへ配分されるように，互いに素な部分集合の合併へデータを分ける方法である．各データの単一コピーだけがシステムに存在して，任意の時点でそれぞれの根拠地となるノードに配置される．一般に，分割の目的は，レコードをもっとも頻繁にアクセスするノードへそのレコードを配置して，応答時間と通信量を最少化することである．データの複製とは，同一のデータの2つ以上のコピーをシステム内に保持し，極端な場合には，全ノードにデータのコピーを持たせる方法である．データ複製の利点は性能，費用，適合性，信頼性を本質的に改善できるということである．しかし，この特長はデータのいくつかのコピー間の無矛盾性を維持するという非常に複雑な更新の犠牲の結果得られる．複製の利点から，複製データの更新問題の研究がかなり注目されており，以下で，さらに詳しく扱うことにする．データの無矛盾性を維持するのに用いる算法では，更新に要する長い遅延を除くために，ノード間の広範な通信が必要となり，性能上重要な問題を起こす可能性がある．最適性能を実現するのは複雑な問題であって，脱漏率，更新頻度，ノード数，データ量，あるいは通信の速度と費用に依存する．

少量のデータを分散させる最適の方法は，更新頻度が低ければ，単純にデータを複製することである．この場合，更新が発生したら後述する同期算法の1つを使って，コピーを持っているすべての場所に更新要求を送る．この方法は更新が束ねられており，かつ，性能の高さと信頼性が重要である場合にはとくに魅力的である．

一方，データ量が少なく更新頻度が高ければ，通信の負荷が重くなるので複製は好ましくなくなる．分割法は脱漏率(非局所アクセス比率)さえ低ければ，最良の方法である．なぜならば更新率が高くても，分割の場合はほとんどの更新が局所的になるからである．また，分割では単一コピーしか存在しないから，遠隔の分割部分への更新に要するオーバ・ヘッドも通信量も遠隔の複製データの更新より少ない．もし脱漏率も更新頻度も高率ならば，データ量にかかわらず集中化するのが最良の方法である．それ以外の場合では，分割が最良である．分割と複製は極端な純粋方法を表しており，実際には，同一のデータに対して両方法を任意に混合して利用している．

また，開発中あるいは使用中の多数のオペレーティング・システムにおいて，分散システムの利用者に論理的に集中したインタフェースを設ける方向がとられている．とくに National Software Works[20]，標準局の Network Access Machine[18]，および ARPANET の RSEXEC[22,23] プロジェクトがあげられる．これらのプロジェクトは，利用者に無矛盾で論理的に集中した使いやすい計算回線網資源を供給するという共通の目的を持っている．この場合，アクセスはシステムに常に独立しているとはいえないが，地理的場所には独立である．

4.4 ディレクトリ

データをシステム全体に分散できるようにするため，データの位置を定めるのに必要なシステムのデータ・ディレクトリを分散することもできる．また，データを分割・複製して冗長度を持たすことができるように，ディレクトリも同様に取り扱うことができる．さらにディレクトリの集中化も可能であろう．データの分散の2つの純粋な方法（分割・複製），およびディレクトリの分散の3つの純粋な方法（分割・複製・集中）を組み合わせる6つの組合せがある．理想的には，汎用分散システムはこの6つの純粋な組合せのすべてや，他のあらゆる方法の混合を許し，システム生成時に最終的な選択が行えるのが望ましい．ディレクトリ自体も1つの重要な相違点を除いてデータと同様に扱うことができる．その相違点とはディレクトリの位置情報がシステムのノードで，あらかじめわかっていなければならないということである．この相違点を実現するには，ディレクトリの移動はごくまれかまったく起こらず，一方，ディレクトリの検索頻度は確かに高いので，各ノードにディレクトリのコピーを蓄えればよい．ディレクトリを通常のデータとして扱うことには他の利点がある．すなわち，記憶域管理・回復・機密保護，あるいは更新の同期のような正規のシステム機能がディレクトリに対して自動的に利用できる点である．

4.5 複数データ・コピーの更新の同期

データベースの完全性を維持するには，更新の同期という非常にむずかしい問題があり，複製あるいは冗長度を持つ分割データベース・システムの特長から，その解決法が研究されている．多くの応用では更新を一括して行えば十分である．しかし，複数コピーを持つデータの実時間更新には多くの問題がある．たとえば，各ノードでは予測不能な遅れの後に初めて他のノードで発生した更新が認知できるので，この遅れを制御しないと，データの内部的な無矛盾性が破られることになる．

集中あるいは冗長度のないデータベースにおいては，ロックのような仕組みをセットして更新を整列させることができる[9,12]．この更新の整列によりデータの無矛盾性が保障できる．複数コピーがあるときにデータの無矛盾性を維持するには，更新を整列して，この更新列をすべてのコピーに同順で適用する必要がある．

2つのレベルの無矛盾性は，データの複数コピーに対して定義できる．すなわち強無矛盾性とは同一時点ですべてのコピーが同一の更新状態にあることであり，弱無矛盾性とは，任意時点ではコピー間で更新状態の遅れが見られるが，時間をかけると同一の更新状態に達することが定義できる．この際，任意時点におけるデータの複数コピー間の差異の尺度として，さらに首尾一貫性 (coherence) が定義できる．また，更新の完了に要する平均遅延時間によってシステムの敏速性 (promptness) が定義できる．

首尾一貫性と敏速性の形式的定義は E. Gelenbe ら[10]により与えられている．分散データベースでは，首尾一貫性と敏速性を同時に高めることはできないという基本的相補性があり，物理学の W. K. Heisenberg の不確定性原理に似ている．

弱無矛盾性の条件のもとで同時並行更新を制御する多くの算法が提案されている[10]．もっとも単純なものは，すべての更新要求を1つの主ノードへ集めて，そこで整列し更新をもとのノードへ送った後に，この順で処理する方法である[1,11]．他の更新制御法は1つの時計を設けて更新の受付時に発生ノードも時刻スタンプを押す方法に基づいている．任意の1ノードが時間変化ごとに，たかだか1つの更新を生成し，ノード番号を時刻スタンプに追加するならば，スタンプを広域的に一意にでき，更新の整列に使用できる．この方法の1つに，“多数決原理”による値に基づくものもある[24]．また時刻スタンプを使う他

の方法として，更新によって変えられる変数で類別するものもある[19]．これらの方法や目下研究中の方法ではデータの完全性が保たれる．その他の目標として，遅れの最小化，ノードの優先的取扱いの排除，更新の取消しの最少化，通信費用の最少化を考慮して，最適な新しい算法を導く必要がある．

4.6 回復の問題

冗長度のない分割データベースでは，ノードの故障に対する回復問題は集中システムの場合と論理的に一致し，その回復技法も似ている．唯一の相異点は広域的要求の存在であるが，これは物理的相異であって論理的相異ではない．オーディット・トレイルとチェックポイント・ダンプが引き続き使用される．分割データベース・システムでの通信の故障においては，発信ノードがメッセージの受取り確認を受信し損うことがあり，回復の負荷は発信ノードに集まる．また，アクセス・トランザクションを通信が修復するまで待たせねばならない．

複製データベースでは，事情はまったく異なる．通信線やノードが故障しても，残りのシステムは正規の動作が続行可能で，故障したノードあるいは通信線にかかわるトランザクションが停止するにとどまる．冗長度のあるデータベースにおいて，回復を実施する原型のいくつかを G. Lelann[15]が論じている．

5. データベース・コンピュータの構造

5.1 連想記憶

データベース・コンピュータの概念における1つの基本は“連想記憶”あるいは“探索記憶”である．連想能力を持つ第1の記憶装置に，マイクロ秒のオーダで一度に数キロバイトの固体記憶域を並行に探索するものがある．これは“並行連想記憶”と呼ばれ，STARAN 計算機[3]がその特徴を表している．

第2の連想記憶は，ディスク，バブル素子等の順記憶域を用いて，順に探索する方法をとる，いわば“順連想記憶”である．典型例では10ミリ秒程度で数メガバイトを探索することができる．初期の順連想記憶の1つに CASSM プロジェクトで使われた例がある[8]．

5.2 データベース・コンピュータの構造

1970年代の始めに，データベースの効率を改善するため，以下の提案がなされた．

1) 既存のミニコンピュータを使うバックエンド・システム
2) 自己管理を行う記憶域階層
3) インテリジェント制御装置
4) データベース・コンピュータ

おそらく，もっとも早いものは伝統的なミニコンピュータに実装したバックエンド・プロセッサである[6]．目的は性能の改善と互換性のないシステム間でデータベースを分かち合うことであった．その後，多数のバックエンド・プロセッサが商品化されている．T. A. Welch[25]が述べる記憶域階層の発想もあり，1970年代の中頃に初めて実装された自己管理記憶域階層は，ファイルとデータの問題を解く1つの試みであった．最近の応用ではキャッシュ・ディスクで，ディスクに対するキャッシュとして半導体記憶素子を使用している．シミュレータ実験で4メガバイトのキャッシュに対する平均ヒット率は75パーセントであったが，これは平均アクセス時間を半減するのに十分な値である．

これまで採用された改良法は，すべて伝統的なホストに入出力チャネルか通信線を介して“バックエンド”をつける形式であり，すべてインテリジェント制御の方法と考えられ

る．これらの構築は平担なファイル構造を使い，情報を内容によって探索するという点で，間接的に，リレーショナルな性格を持つ．この種の構築の初期の例に CAFS システムがある．CAFS は RARES の拡張で，固定ヘッド・ディスクを基本とし，単一のパイプライン命令流れを使用している．CAFS は内容による番地づけの最初の試みであって，基準システムと考えられる．CAFS では十分なバッファリングも並行処理も行っていない．

リレーショナル連想プロセッサ[21] RAP では固体素子の処理要素の配列を使用する．この各処理要素は全データベースの各分割を処理する．

RAP では並列処理が導入され，それによってデータベースを分割して局所的な記憶装置の集合に写像している．

データベース・プロセッサへの，やや異質なアプローチに STARAN 計算機[3]の使用があった．伝統的記憶装置の中のデータはこの場合，連想レジスタ配列に送られ特定のキーの値に対する探索を行う．STARAN のアプローチは概念的に CAFS に類似しており，ただ固定ヘッド・ディスクはランダム・アクセス主記憶に置き換えられ，単純な比較論理は大きな並列連想記憶配列に置き換えられている．重要な相異は STARAN 計算機では 1 パスで単一ファイルに対する複数路の探索を実行できる点である．

これら CAFS, RAP, STARAN のアプローチの 3 つの方法および類似の方法はデータベースの性能を改善するが，次の欠点を持っている．

1) 大容量記憶域が高価なため，サイズが限定されること
2) データベース全体を大容量記憶域に常駐させねばならないこと
3) 完全なリレーショナル・ジョインが実行できないこと
4) 分類が実行できないこと

データベース構築の進展の次の段階は，データベース・プロセッサからデータベース・コンピュータ[14]への移行であった．大容量記憶域の媒体は移動ヘッド・ディスクで，固定ヘッド・ディスクよりビット当たりの貯蔵費用が低い．データベースをシリンダの集合に分割する考え方が導入され，(いくつかの可能な) アクセス・フィールドのハッシュを行って探索するシリンダを選ぶ方法をとる．ディスクの 1 トラック分の情報を完全にバッファリングし，RAP と同様の方法で並行処理を用いて，同時に数トラックからのデータを処理する．

最近 D. K. Hsiao のデータベース・コンピュータの概念は，さらに拡張された．設計目的は変わらず，100メガバイトから100ギガバイトのオーダのネットワーク，リレーショナルあるいはハイラキカル・データベースの支援である．設計法も不変である．すなわち，処理要素の配列を制御する汎用コンピュータを持つバックエンド・コンピュータとインテリジェント機能を持つ並列移送ディスク制御と並列移送ディスクによる．プロセッサ間の限定した通信量を加えてジョイン操作と分類を行えるように拡張された．この通信では，隣接プロセッサが同一バッファ記憶モジュールをアクセスできることが必要である．

また，低費用の固体素子の順連想記憶が付加され，一意キーと分類フィールドの値を固定するのに使われる．この結果として，データベース・コンピュータ構築は図 3 のように次の 6 つの主要素から形成される．

1) プロセッサ要素
2) データベース・プロセッサ制御
3) トラック・バッファ
4) 連想記憶域

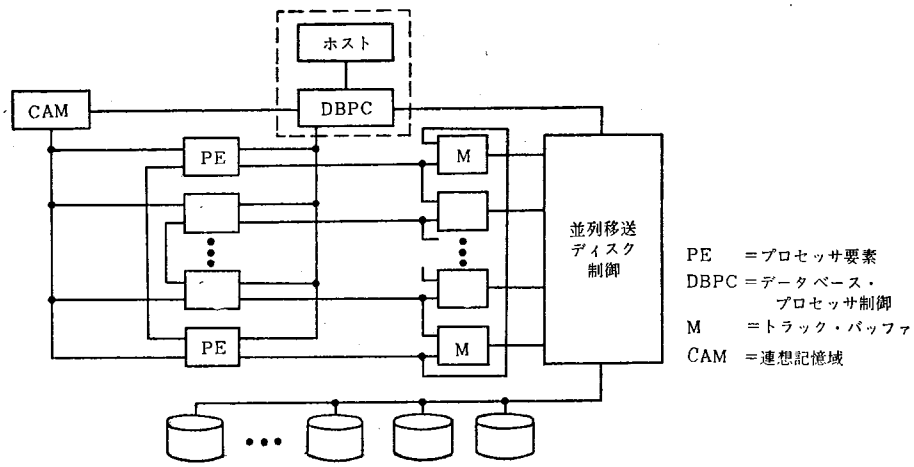


図 3　データベース・コンピュータの構成
Fig. 3　Data base computer organization

5)　並列移送ディスク

6)　並列移送ディスク制御

たとえば，ホストはネットワークあるいはリレーショナル・モデルの高水準言語命令を使ってデータベース・コンピュータとインタフェースする．これらの命令はデータベース・プロセッサ制御に送られて解析される．この解析に基づいて適当なパラメタの集合がつくられ，プロセッサ要素へと移される．データベース・プロセッサ制御は，また並列移送ディスクに対する必要な入出力の命令を生成する．

並列移送ディスクでは，同時に1シリンダ内のすべてのトラックを読むことができて，各トラックの情報はそれぞれのトラック・バッファに移送される．各トラック・バッファは2つの隣り合うプロセッサで使うことができ，それぞれ4トラック分の情報が蓄えられる大きさを持っている．トラック・バッファにロードされると，プロセッサ要素は非同期的にデータの操作を開始し，選択，射影，結合あるいは分類等を実行する．基準に合ったデータベース・プロセッサ制御にもどされ，次にホストへもどされる．リレーショナル・データベース・システムを自明な方法で支援でき，またネットワーク・データベース・システムも支援できる．

これらの装置は互いに共同して動作する．シリンダ内のすべてのトラックを並列的に探索できるように，探索パラメタはプロセッサ要素にロードされる．

トラック・バッファのサイズは二重バッファリングを行うのに十分な大きさであり，これらのトラック・バッファの動作はプロセッサ要素が制御する．プロセッサ要素は十分に強力であり，(既存機械で行ったような) 問合せの中のパラメタによる等号での突合せだけでなく，$>$, $<$, $\neq$, AND, OR, あるいは NOT の機能を実行する．

バブル素子記憶の新しい固体直接アクセス技術は，オーダで1ないし2遅いのでトラック・バッファの要件に合わない．最近の調査によれば典型的レコード・サイズは約200バイト長であった．探索で，もっとも一般的な操作はキー・フィールドの最初の数バイトを見て該当するレコードか否かを見ることで，もし該当しなければ200バイトをスキップして次のレコードに飛ぶ．RAM ならば，これを1マイクロ秒程度の時間で一時に行うこと

ができる.

5.3 データベース・コンピュータの応用

データベース・コンピュータはデータベース情報のアクセスと処理の効率を本質的に改善するが，現在の技術より万能的によいとはいえない．このことは重要な点である．データベース・コンピュータがより効果的になるためには，次の2つの条件が必要である．

1) データベースの応用が全システム負荷の相当部分を占めること
2) データ・アクセスが，伝統的技法でいくつかの大容量記憶域を使用するものであること

多数の実験や性能評価によると，データベース・コンピュータはこれら2つの基準に合う場合にはオーダが5ないし50倍の性能改善を行いうるが，これらの基準に合わない分野では，効率を損うだけである．

6. おわりに

利用者の需要と進展する技術により，データベース技術は論理的，物理的およびシステム構築のレベルで著しく進歩している．物理的装置の費用対性能比は2～3年ごとに2倍の割合で進んでいる．ネットワークとリレーショナル・システム間のスキーマ変換を実行可能にする共通データモデルが開発され，論理レベルでも著しい進歩がみられる．共通データモデルはポインタを基本とする記憶構造と，値を基本とする構造の双方がネットワーク，あるいはリレーショナルのいずれの利用者インタフェースをも支援できることを示している．これは単一データベース構造の複数モデルの支援ができることも意味する．

論理および物理レベルの進展によって，データベース・コンピュータに象徴される構築レベルの改良は比較的穏当な費用で性能を著しく向上させることを保証する．したがって，費用面，性能面および利用者インタフェースの面で著しい改善が得られる．一般データベース・システムの将来は明るく，個々の利用者が“固有システムを構成する”必要性を取り除くことができるにちがいない．

(1100 ソフトウェア開発部 原 潔 訳)

参考文献 [1] P. A. Alsberg and J. D. Day, “A Principle for Resilient Sharing of Distributed Resources,” *Proc. 2nd Int'l Conf. Software Engineering*, San Francisco, Oct. 1976, pp. 562-570.

[2] M. M. Astrahan, et al., “System R, A Relational Data Base Management System,” *Computer*, Vol, 12, No. 5, May 1979, pp. 42-48.

[3] P. B. Berra and A. K. Singhania, “A Multiple Associative Memory Organization for Pipelining a Directory to a Very Large Data Base,” *Digest of Papers*, COMPCON 76 Spring, San Francisco, Feb. 1976, pp. 109-112.

[4] A. H. Bobeck, P. I. Bonyhard and J. E. Geusic, “Magnetic Bubbles—An Emerging New Memory Technology,” *Proc. IEEE*, Vol. 63, No. 8, Aug. 1975, pp. 1176-1195.

[5] G. M. Booth, “Distributed Information Systems,” *AFIPS Conf. Proc.*, Vol. 45, 1976 NCC, pp. 789-795.

[6] R. H. Canaday, et al., “A Back-end Computer for Data Base Management,” *Comm. ACM*, Vol. 17, No. 10, Oct. 1974, pp. 575-582.

[7] G. A. Champine, “Four Approaches to a Data Base Computer,” *Datamation*, Vol. 24, No. 13, Dec. 1978, pp. 100-106.

[8] G. P. Copeland, G. J. Lipovski and S. Y. W. Su, “The Architecture of CASSM : A Cellular System for Nonnumeric Processing,” *Proc. 1st Annual Symposium Computer Architecture*, Dec. 1973, pp. 121-128.

[9] K. P. Eswaran, J. N. Gray, R. A. Lorie and I. L. Traiger, “The Notions of Consistency and Predicate Locks in a Database System,” *Comm. ACM*, Vol. 19, No. 11, Nov. 1976,

pp. 624-633.

[10] E. Gelenbe and J. Sevcik, "Analysis of Update Synchronization for Multiple Copy Data Bases," *Proc. 3rd Berkeley Workshop on Distributed Data Management and Computer Networks*, Aug. 1978, pp. 69-90.

[11] E. Grapa and G. G. Belford, *Techniques for Update Synchronization in Distributed Data Bases*, Center for Advanced Computation Report, Univ. of Illinois, 1977.

[12] J. N. Gray, R. A. Lorie, G. R. Putzolu and I. L. Traiger, *Granularity of Locks and Degrees of Consistency in a Shared Database*, IBM San Jose Laboratory Report, 1975.

[13] A. S. Hoagland, "Storage Technology : Capabilities and Limitations," *Computer*, Vol. 12, No. 5, May 1979, pp. 12-18.

[14] D. K. Hsiao and S. E. Madnick, "Database Machine Architecture in the Context of Information Technology Evolution," *Proc. 3rd Int'l Conf. Very Large Data Bases*, Tokyo, Oct. 1977, pp. 63-84.

[15] G. Lelann, "Algorithms for Distributed Data Sharing Systems Which Use Tickets," *Proc. 3rd Berkeley Workshop on Distributed Data Management and Computer Networks*, Aug. 1978, pp. 259-272.

[16] S. C. Lin, D. C. P. Smith and J. M. Smith, "The Design of a Rotating Associative Memory for Relational Database Applications," *ACM Trans. Database Systems*, Vol. 1, No. 1, Mar. 1976, pp. 53-75.

[17] G. Panigrahi, "Charge-Coupled Memories for Computer Systems," *Computer*, Vol. 9, No. 4, Apr. 1976, pp. 33-42.

[18] R. Rosenthal, *A Review of Network Access Techniques with a Case Study: The Networks Access Machine*, NBS Technical Note 917, July 1976.

[19] J. B. Rothnie, N. Goodman, P. A. Bernstein and C. A. Papadimitriou, *The Redundant Update Methodology of SSD-1: A System for Distributed Databases*, Technical Report No. CCA-77-02, Computer Corporation of America, Feb. 1977.

[20] R. E. Schantz and R. E. Millstein, *The FOREMAN: Providing the Program Execution Environment for the National Software Works*, Report No. 3266, Bolt Beranek and Newman, Inc., Mar. 1976.

[21] E. A. Ozkarahan, S. A. Schuster and K. C. Smith, "RAP—An Associative Processor for Data Base Management," *AFIPS Conf. Proc.*, Vol. 44, 1975 NCC, pp. 379-387.

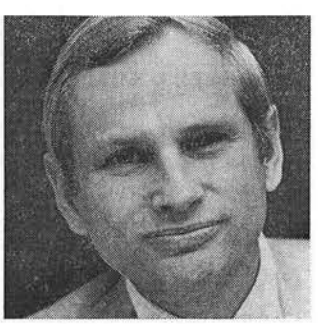
[22] R. H. Thomas, "A Resource Sharing Executive for the Arpanet," *AFIPS Conf. Proc.*, Vol. 42, 1973 NCC, pp. 155-163.

[23] R. H. Thomas, *A Solution to the Update Problem for Multiple Copy Data Bases Which Use Distributed Control*, Report No. 3340, Bolt Beranek and Newman, Inc., July 1975.

[24] R. H. Thomas, *A Majority Consensus Approach to Concurrency Control for Multiple Data Bases*, Report No. 3733, Bolt Beranek and Newman, Inc., Dec. 1977.

[25] T. A. Welch, "Memory Hierarchy Configuration Analysis," *IEEE Trans. Comput.* Vol. C-27, No. 5, May 1978, pp. 408-413. (本誌 pp. 43-51 参照)

執筆者紹介 G. A. シャンパイン (George A. Champine)



Sperry Univac 社の大型商用コンピュータ・システムのアドバンスト・システム部長を務め，他にシステムの企画・設計・分析を担当．商用機器を応用した大規模な特殊プロジェクトにも従事．同社在籍の20年間にソフトウェアやシステム設計の分野で技術的・管理的ポジションを歴任．上級スタッフ・コンサルタントで大規模コンピュータ・システムの先進技術プログラムの管理を担当．また Minnesota 大学の助教授を兼任し，アドバンスト・インフォメーション・システムの設計分析ならびに技術プロジェクト管理を指導．同大学において物理学で BS と MS を，経営情報システムで Ph. D を取得．現在，Vydec 社副社長．

記憶域階層構成の解析

Memory Hierarchy Configuration Anarysis

T. A. Welch

要 約 本稿では，記憶域階層設計における速度と費用のトレード・オフを解析的に述べる．与えられた条件(記憶域サイズ，アクセス確率を持つ階層に対する費用等)の下で，平均アクセス時間，すなわち記憶システム遅延を最小化するような最適化基準を考える．記憶装置の速度と費用を関係づける“ベキ関数”の仮定を用い，最適化された階層では，費用と遅延の均衡配分特性があることを明らかにする．この場合，各記憶装置に投資するシステム費用の割合は，その装置によって引き起こされる平均アクセス遅延の割合に等しい．同じ仮定を用いて，平均アクセス時間の下限が求められる．これに基づき大雑把にいうと，アクセス時間はアクセス確率の立方根で表される．これらの結果は記憶域階層設計の方策を展開したり，データ配置算法を評価する際の有用な道具となる．

Abstract This paper presents an analytical study of speed-cost tradeoffs in memory hierarchy design. It develops an optimization criterion by which average access time, i.e., memory system delay, is minimized under a cost constraint for a hierarchy with given memory sizes and access probabilities. Using a power function assumption relating speed and cost of memory units, it is shown that an optimized hierarchy has the property of balanced cost and delay distributions, in that each memory unit makes the same percentage contribution to memory system cost as it makes to average system access delay. Using the same assumption, a lower bound on average access time is developed, showing that access time is roughly related to a cube-root averaging of access probabilities. These results provide useful tools for developing memory hierarchy design strategies and for evaluating data placement algorithms.

1. はじめに

多様な新しい記憶域技術の発展につれ，記憶域階層の概念はコンピュータ・システムのサイズによらず一層重要となりつつある．ミニコンピュータにおいても近い将来の典型的構成は，4種以上の異なった速度の記憶域を持つと予想される．また六階層記憶域システムは大きなコンピュータ・システムにおいて現在特別なものではない(たとえばレジスタ，キャッシュ，コア，ドラム，ディスク，テープ)．コンピュータ・システム設計で経済的な意味で重要な問題は，これら多様な記憶域の相対速度とサイズを選択して，費用と平均データ・アクセス遅延の双方を最小化することである．多段階層に対する解析手段はその重要性により，いろいろな方向[1-4]から研究がなされている．本稿ではC. K. Chow[1]の方法と目標が似ている単純な解析的方法を展開する．記憶域の費用とアクセス時間の解析的モデルを利用した定量的方法で展開し，記憶域階層構成のトレード・オフの問題が一層解明されるが，データ配置問題(ページ・サイズ，記憶域配置，置換え算法等)を直接扱うことはせずに，記憶装置のサイズ，速度，数についての決定に限ることとする．

まず，固定サイズの記憶域の集合について解析をはじめ，与えられた全費用でシステムの平均アクセス時間を最小化するように，記憶装置の速度と費用を選択する基準を展開す

る．そして，ある機器の費用の条件の下で，各記憶装置がその費用に比例するアクセス遅延を示すとき，システムは最小のアクセス時間を実現することを示す．このことから派生して，システム全体の費用が決められている場合，記憶装置の速度の組合せを問わなくとも，与えられたアプリケーションに対してその費用で達しうる平均アクセス遅延時間の最小限界を求めることができる．この応用として，一定費用の記憶階層ではアクセス確率に大きな片寄りがある場合に限り，つまり全アクセスの大部分が一部の記憶域に集中しているときにのみ，階層の追加が効率改善に著しい効果をもたらすことを例にあげて示す．

2. 記憶システムのモデル

N個の記憶装置 $M_i(1 \leq i \leq N)$ からなる記憶域階層を考える．M_1 を最小でもっとも速い記憶域とし，M_n を最大でもっとも遅いとする．各装置は，C. K. Chow[1] の記法に従い，次のパラメタで特徴づけられる．

B_i： M_i の費用

b_i： M_i の1データ単位の費用（例：バイト当たりの費用）

c_i： データ単位で表した M_i のサイズ；$B_i = b_i c_i$

t_i： M_i の1データ単位への平均アクセス時間；詳しくは，計算が M_i からの応答を待つ時間の遅れ

P_i： 特定のデータ参照が M_i にアクセスを要求する確率

p_i： M_i の1データ単位がアクセスされる平均確率；$P_i = p_i c_i$

また，システムのパラメタは

$S = \sum B_i$：記憶域システムの費用（N個の記憶域について和をとる）

$T_{avg} = \sum t_i P_i$：システムの平均アクセス時間（単一プロセッサ・システムを仮定する）

となる．

記憶域階層設計の一般的な目的は，N, t_i, c_i を調節して，与えられた費用 S に対して最小の平均アクセス時間 T_{avg} を実現することである．逆に与えられた T_{avg} に対して，S を最小化する変形問題は同じ数学を使って同一形の結果が得られる．したがって，両者を分けて考えない．T_{avg} の最小化は，次の2つの関数の解析を必要とする．

1) 機器関数：記憶域機器における t_i と b_i とを関係づける関数

2) 利用関数：記憶域システムへのアクセス分布を示し，P_i と c_i を関係づける関数

本稿では記憶装置の数とサイズを固定し，P_i を与えられたとして，利用関数は論じない．

単純にするため，各記憶装置では直接の処理を仮定する．（たとえば，命令はディスクの制御から抜けたら，即実行できるとする．）また，記憶域間のデータ移送に要する時間は計算とは干渉しないと仮定する．異なったシステムについても，一般モデルの妥当性をくずさないように，p_i と t_i を正しく選択してモデル化できるものとする．その他の仮定については，個別のモデルを検討する際に導入することにする．

利用可能な記憶装置のアクセス時間に対する装置費用をプロットすると，機器モデルは1つの単調減少曲線となる．すなわち，速い装置は高い．装置の速度を連続関数と仮定して，この曲線を解析的関数で近似すると便利である．この仮定はもちろん妥当ではないが，実際の装置を使って得られる性能の指針と限界となる関係を導くのに非常に助けとなる．

機器関数の適当なモデルはベキ関数 $b(t) = b_0 t^{-\beta}, t > 0$ である．β の典型的な値は0.5の近傍である．たとえば，記憶域 X が記憶域 Y の4倍の速さならば，ビット当たり費用は

2倍であることを示す．ベキ関数を使うことの妥当性は時間範囲で，8ぐらいにわたる実際のデータに大雑把に当てはまるからである．S. L. Rege のデータ[4]は，たとえば $\beta=0.5$ についてこれを示し，装置費用の代わりにシステム費用を使うと，実際のデータとよく一致する．実際のデータとはいえ，システム費用対装置費用，システム費用対システム価格，めまぐるしい技術変化，アクセス時間が一定でないディスクのような装置に対するアクセス時間の見積り方，等による曖昧さから不正確さが残る．したがって b_0 と β は，その解析の対象となるシステムのアプリケーションにより，広範に変化する量とみるのが妥当である．

3. 記憶域構成の最適化

最適化定理　おのおのがサイズ t_i とアクセス確率 P_i を持つ記憶域 M_i の集合を考える．さらに機器関数 $B_i=f(t_i)$ を仮定して，各記憶域の費用 B_i とアクセス時間 t_i を調整できるとする．このとき，費用の制約を，

$$\sum B_i \leq \text{一定}$$

と仮定すれば，

$$P_i=-K\frac{dB_i}{dt_i}, \qquad K=\text{一定} \tag{3-1}$$

のとき，平均アクセス時間 $T_{avg}=\sum P_i t_i$ は最小となる．

証明……最小の T_{avg} において，B_i と t_i の増分 dB_i と dt_i は次の2つの条件を満たさなければならない．

$$\sum dB_i \leq 0 \quad \text{(費用は増加しない)} \tag{3-2}$$

$$dT_{avg}=\sum P_i dt_i \geq 0 \quad \text{(これより小さい } T_{avg} \text{ は存在しない)} \tag{3-3}$$

各記憶域 M_i は勾配 $dB_i/dt_i=-z_i$ を持つ機器曲線上の点をとるから，これを dt_i に代入すると (3-2), (3-3) は，(t_i の微小な変化では z_i を一定と仮定する.)

$$\sum dB_i \leq 0 \tag{3-4}$$

$$\sum \frac{P_i}{-z_i} dB_i \geq 0 \tag{3-5}$$

となる．もし，$P_x/z_x>P_y/z_y$ となる M_x と M_y が存在したとして，$dB_x=\delta>0$, $dB_y=-\delta$, その他の i について $dB_i=0$ とすると，(3-4) は満たすが (3-5) は満たさない．したがって，(3-4), (3-5) を満足するためには，すべての P_i/z_i は同一の値をとらなければならない．すなわち定数 K があって，すべての M_i に対して，$P_i/-z_i=K$ である．よって，最小の T_{avg} において，$P_i=-K(dB_i/dt_i)$ となる．

解釈……この基本定理から直接直観的な見通しが得られるわけではないが，P_i の関数として(ふつうは唯一の)最適な B_i を導く手段が与えられる．しかし，一般のアプリケーションに対して定数 K を評価するのは容易ではない．そこで，この結果を役立てるには，さらに解析を進める必要がある．とくに，特定の機器関数を使うと興味ある結果が得られる．

4. 機器に対するベキ関数モデル

あらかじめ特定のコンピュータ設計に利用可能な記憶機器は，$b_i=b_0 t_i^{-\beta}$ で表現されるとする．この仮定の下では，次に示すように単純かつ強力な階層最適化関係が存在する．これは，特定の利用関数と特定の階層構成に対して，C. K. Chow[1] が最初に観察した関係の一般化である．

記憶域均衡定理　各 M_i について P_i がわかっているような固定サイズの記憶域階層が与えられ，各機器は $b_i=b_0t_i^{-\beta}$ で表されるとする．そのとき個々の記憶域速度を一定費用条件 $\sum B_i=S$ の下で T_{avg} を最小にするように選ぶとすると，

$$\frac{P_it_i}{T_{avg}}=\frac{B_i}{S}$$

すなわち，M_i へ投資するシステム費用の割合は M_i によるシステム遅延の割合に等しくなる．

証明……サイズ c_i の記憶域に対して $B_i=c_ib_i$ で，$b_i=b_0t_i^{-\beta}$ を使うと，

$$\frac{dB_i}{dt_i}=-\beta b_0c_it_i^{-\beta-1}=-\beta\frac{B_i}{t_i}$$

となる．最適化定理を使って，T_{avg} が最小となるのは，

$$P_i=-K\frac{dB_i}{dt_i}=\beta K\frac{B_i}{t_i},\ \text{すなわち，}\ P_it_i=\beta KB_i \tag{4-1}$$

である．すべての M_i について和をとると，

$$\sum P_it_i=T_{avg}=\sum\beta KB_i=\beta KS,\ \text{すなわち，}\ K=\frac{T_{avg}}{\beta S} \tag{4-2}$$

となる．(4-2) を (4-1) に代入して，結果を得る．

解釈……この定理は記憶域階層解析の非常に便利な道具となる．ある記憶域装置がそのシステム費用に対する割合よりもシステム・アクセス遅延に対して高い割合を示すならば，(より速い装置を使うか，余分のアクセス・チャネルを付加するかなどにより)速度を上げるべきであることを示している．いいかえると，遅延時間より相対的に多い費用を示す記憶装置は遅くし，節約できた金額を階層の他の記憶域に使う方がよいことを示している．この定理は任意の記憶域階層構成と任意の利用関数に対して成立する．しかし，次の2つの理由から，注意深くこれを解釈しなければならない．

1) この定理は記憶域アクセス時間の変化にのみ適合し，記憶域アクセス確率の変化には適用できない．したがって，多様な記憶域へのデータの配置変更にも当てはまらない．記憶域の費用と利用の均衡は1つの最適システム設計を確定するものではなく，与えられたデータ配置の集合に対して記憶装置のアクセス時間を選ぶとき，定理に示された選択が最善のものであることを保障するにすぎない．
2) アクセス時間は注意深く定義しなければならない．たとえば，ある多重処理システムでシステムのスループットを最大にしようとするならば，記憶装置に対するアクセス時間は命令の有効な実行にあたって空費した時間だけを意味することになる．遅い装置をアクセスするときに遅れる時間は，たとえばプロセスへスイッチするための負荷だけかもしれない．

実際の機器は $b_i=b_0t_i^{-\beta}$ 曲線に正確に当てはまらないので，均衡関係は記憶システムの微調整には通常役立たない．しかし，一般のシステム解析，とくに初期設計時には非常に役立つであろう．

例 1　データの10パーセントの部分へアクセスの90パーセントがなされるようなデータ集合を仮定する．表1は10パーセントのデータを速い記憶域へ蓄えた二階層記憶域システムにおいて，アクセス時間を比較したものである．表の左第1欄，第2欄では，ある費用配分を仮定した．次に $\beta=0.5$ を使ってアクセス時間 t_1, t_2 を導いた．さらに T_{avg} は $P_1t_1+P_2t_2$ から算出した．P_1t_1/T_{avg} の欄はこの計量を B_1/S と比較するために含めてあり，T_{avg} が最小のとき一致している．

便宜上，費用とアクセス時間は，システム費用が1となり，一階層記憶域システムのアクセス時間が1となるように正規化してある．この例では，$P_1/P_2=90/10$ の利用関数に対して二階層記憶域システムでは，一階層記憶域システムのほぼ4倍速くすることができることを示している．

表 1 二階層記憶域システムの性能
Table 1 Performance of two-memory systems

B_1	B_2	t_1	t_2	T_{avg}	P_1t_1/T_{avg}	P_2t_2/T_{avg}
0.1	0.9	1	1	1	0.900	0.100
0.2	0.8	0.250	1.27	0.352	0.640	0.360
0.3	0.7	0.111	1.65	0.265	0.377	0.623
0.325	0.675	0.095	1.78	0.263	0.324	0.676
0.4	0.6	0.063	2.25	0.281	0.200	0.800
0.5	0.5	0.040	3.24	0.360	0.100	0.900
費用配分		記憶域アクセス時間		システムの有効アクセス時間	遅延の配分	

ただし，$P_1=0.9$, $P_2=0.1$, $c_1=0.1$, $c_2=0.9$, $\beta=0.5$, $b_0=1$, $S=1$.

増分解析 同様の推論によって，記憶域の個別域に対するアクセス度数の関数として，最良アクセス時間を評価できる．

個別の記憶域に対して (4-1) を使って，$P_i=\beta K(B_i/t_i)$ と $b_i=b_0t_i^{-\beta}$ と $B_i=b_ic_i$ を結びつけると，

$$t_i=(\beta Kb_0c_i)^{1/(\beta+1)}P_i^{-1/(\beta+1)}$$
$$b_i=b_0^{1/(\beta+1)}(\beta Kc_i)^{-\beta/(\beta+1)}P_i^{\beta/(\beta+1)}. \tag{4-3}$$

$\beta=0.5$ の典型例に対して，データ単位当たりの平均アクセス確率 $p_i=P_i/c_i$ を使うと，

$$t_i=K'p_i^{-2/3}, \qquad b_i=K''p_i^{1/3}. \tag{4-4}$$

これは，たとえば一方が他方より2倍もアクセスされるような2つの記憶場所があるとき，T_{avg} を最小にするには，アクセス度数の多いデータを小さいデータの $2^{0.67}=1.59$ 倍速く，ビット当たり $2^{0.33}=1.26$ 倍高い記憶域へ蓄えるべきであることを意味している．

ほとんどのコンピュータ設計では $\beta=0.5$ 以外の機器を使用するが，すべての適切な β の値は上の関係において，やはり小さい指数となる．指数が小さければ，記憶階層における記憶域の速度と費用の組合せの使用を保証するためには，p_i の有意な差を必要とする．

5. 階層の性能の限界

機器関数にベキ関数を仮定すると，特定の利用関数に対しての平均アクセス時間の下限を与える便利な関係が導かれる．この下限は，個別の記憶域のアクセス確率を変えて，システムのアクセス時間に与える効果を調べるのに役立つ．また，どの時点で新たな階層を付加すべきかをおおまかに知ることができる．

性能限界定理 各 M_i に対して p_i, c_i したがって P_i を持つ記憶域階層が与えられ，機器関数 $b_i=b_0t_i^{-\beta}$ を仮定するとき，一定費用制約の下での最小アクセス時間は次式で与えられる．

$$\min(T_{avg})=S^{-1/\beta}b_0^{1/\beta}[\sum c_ip_i^{\beta/(\beta+1)}]^{(1+\beta)/\beta} \tag{5-1}$$

証明……最小の T_{avg} は最適化定理が成り立つときに実現する．それはベキ関数形の機

器関数の場合には，(4-1)の$P_it_i=\beta KB_i$が成り立つときである．与えられた条件$b_i=b_0t_i^{-\beta}$と$B_i=b_ic_i$を使って，$t_i=b_0^{1/\beta}B_i^{-1/\beta}c_i^{1/\beta}$が求められ，これを用いて(4-1)から$t_i$を消去すると，

$$P_ib_0^{1/\beta}c_i^{1/\beta}=\beta KB_iB_i^{1/\beta}$$

となる．B_iを分離して，すべてのiについて和をとると，

$$S=\sum B_i=\sum[b_0^{1/\beta}c_i^{1/\beta}P_i\beta^{-1}K^{-1}]^{\beta/(\beta+1)}$$

$$S=(K\beta)^{-\beta/(\beta+1)}b_0^{1/(1+\beta)}\sum[P_ic_i^{1/\beta}]^{\beta/(\beta+1)}$$

である．そこで，Kについて解き，(4-2)に代入すると，$T_{avg}=K\beta S$であったから

$$T_{avg}=K\beta S=S^{-1/\beta}b_0^{1/\beta}[\sum(P_ic_i^{1/\beta})^{\beta/(\beta+1)}]^{(\beta+1)/\beta} \tag{5-2}$$

となる．$P_i=c_ip_i$を(5-2)に代入すると定理式が得られる．

解釈……この定理は，記憶域階層において記憶域のサイズを変化させたときの効果をあらかじめ評価するときに重要となる．すなわち特定の記憶域サイズの集合に対して可能な最小のT_{avg}(与えられた費用で)を速く評価する手段を与える．また，多段記憶域階層をとるべきデータ・アクセス確率分布についての見通しを得るのにも使える．次の例2および例3でこれらの使用を示す．

通常使用するときは，異なる変数に対して典型的な値を選んで，定理を簡略化できる．記憶域のサイズと費用とをb_0，S，$\sum c_i=1$のように正規化すると使いやすく，このときP_iは確率密度となる．この値の組に対しては，一階層記憶域システムで$T_{avg}=1.0$となる．したがって，他の記憶域構成について計算したT_{avg}は同一費用の一階層記憶域に対する比としてアクセス時間を示す．そこで，便宜上$\beta=0.5$を使うと，定理(5-1)は，

$$\min(T_{avg})=(\sum p_i^{1/3}c_i)^3 \tag{5-3}$$

のようになる．この1/3乗の重みを持つ関数は，T_{avg}に非常に影響するp_iの値が広い変域をとることを示している．

例 2　ディレクトリを持つデータベースを考えよう．ディレクトリは全記憶域の20パーセントを占めるが，そのアクセス時間は全アクセス時間の半分であると仮定しよう．このとき，別の速い記憶域にディレクトリを蓄えるとすると，どれだけ速度は向上するだろうか．(5-3)を使うと，

$$\min(T_{avg})=\left[\left(\frac{0.5}{0.2}\right)^{1/3}0.2+\left(\frac{0.5}{0.8}\right)^{1/3}0.8\right]^3=0.872.$$

したがって，同一費用の一階層記憶域に対して二階層記憶域を使うと，最大約15パーセントの速度改善がなされる．これは，その改良値が一階層の代わりに二階層記憶域を使う場合に増加する仕事量としばしば見合わないことを示す．

一般に「速い記憶域によって，全アクセスの半分が相対的に短いアクセス時間で実行できるから，二階層記憶域システムはほぼ100パーセント近い速い処理が実現できる」と誤解されている．したがって，(5-3)はこの直観と反する．しかし，この15パーセントという値は二階層記憶域システムと，特定の一階層記憶域システム(二階層記憶域システムで付加した速い記憶装置と同額の費用で実現された速度の向上した一階層記憶域システム)とを比較したものである．

この結果はβの値にあまり影響されない．$\beta=0.7$という非常に高い値に対しても，$\min(T_{avg})=0.885$であり，$\beta=0.3$という非常に低い値でも，$\min(T_{avg})=0.856$である．

例 3　一階層記憶域システムに対して二階層記憶域システムを用いてアクセス時間で2倍に改善するのには，どのようなデータ・アクセス分布が必要か．これは非常に興味深

い問題である．本稿で2倍あるいは $T_{avg}=0.5$ という値を典型的改善として選んだのは，異なった記憶域間のインタフェースとか，速い記憶域内に実際に使用するデータを連続的に移すデータ配置算法の実行とかといった複雑さが，なぜ増大するかを完全に正当化するのに必要な値としてである．(5-3) を使うと，

$$T_{avg}=\left[\left(\frac{P_1}{c_1}\right)^{1/3}c_1+\left(\frac{P_2}{c_2}\right)^{1/3}c_2\right]^3=0.5$$

となるが，この等式を満たすサンプル値として，

$$\begin{cases}c_1=0.20\\P_1=0.84\end{cases}\quad\begin{cases}c_1=0.10\\P_1=0.73\end{cases}\quad\begin{cases}c_1=0.05\\P_1=0.66\end{cases}\quad\begin{cases}c_1=0.01\\P_1=0.56\end{cases}$$

を得る．たとえばオペレーティング・システム (OS) のコードの一部が高速の変更可能な制御記憶に蓄えられるような構成を考えるとしよう．上の値を使って，選んだ OS コードが当初全主記憶量の10パーセントを占めるとすると，この二階層記憶域のアプローチが有意の性能改善を示すためには，全体の記憶参照の73パーセントが OS の記憶域 (10パーセント) 部分で費やされることになる．主記憶の 10 パーセント程度しか使用しない OS や実行時間の73パーセントほどを使用する OS はほとんどない．したがって，上記の計算は，特定の記憶装置の組が使えないならば，このアプローチによる大きい性能改善が見込めないことを示している．

6. モデルの解析

いままで使ったモデルは非常に単純なものであった．たとえば，すべての記憶域からの直接処理を仮定し，またいろいろな階層の記憶域でデータは重複しないことを仮定した．

以下ではモデルに誤差の原因を与えて，それらのいくつかを回避する方法を示す．

データ配置　記憶域階層でよい性能を実現するには，まさに使われようとするデータがもっとも速い記憶装置にあるように，データを連続的に置き換える必要がある．このモデルではこの移送が起こり，結果が P_i に反映されると仮定する．データ配置決定の計算とデータを移すのに要する時間はモデルには含めない．そこで，そうしたすべての負荷はこのモデル解析と別に計算しなければならない．もちろん，要求時ページングのような，ある共通の配置算法はほんの少ししか余分な計算とデータ移動時間を要しないから，モデルはこうした場合には直接に適用できる．データ配置算法によって遅延が起こるなら，多階層記憶域システムはよくないとみなされる．これがしばしば多階層記憶域システムの効用を制約する主な要因となる．

機器関数近似　利用可能機器の速度と費用を記述するのに，あるベキ関数を使う点が本稿のもっとも興味深い結果である．前述のように，これは粗い近似にすぎないが，広範囲のアクセス時間にわたって実際のデータに当てはまる関数である．実際のデータがモデル曲線によく当てはまらないならば，モデルの結果は一般のシステム設計の指針として使えるだけで，微調整には使えない．ベキ関数の指数 β はアプリケーションによって変わる．たとえば，記憶域費用の限界として，ベキ関数を使う場合と近似関数として使う場合とは異なる β が必要となろう．このモデルで展開した基本的洞察の多くは，0.3 から 0.7 の大部分の範囲で，β の値にあまり影響されない．$\beta=0.5$ の値は記憶域階層のトレード・オフの実用上の評価に便利である．

多重アクセス　多くのシステムにおいて，二次記憶から検索したデータは使用前に一次記憶へ移さなければならない．こうした場合，一階層以上の記憶域を特定のオペランド

抽出のためにアクセスしなければならない．ある場合には，これら余分のアクセスは計算を遅らせ，ある場合にはこれらは他の演算と並行して実行される．そこで，解析は特定の記憶装置で使われる管理方策に依存する．本稿に展開したモデルで，直接処理（データ参照当たり1記憶域アクセス）を仮定したが，P_i の和が1になる必要がないならば，このモデルは正確に多重アクセスの場合に使うことができる．すなわち，M_i を使うオペランドの割合を表現するものとして，P_i を解釈するわけである．あるオペランドが二階層以上の記憶域をアクセスするならば，$\sum P_i$ は1を越える．本稿で導いた結果には $\sum P_i$ についての条件をつけなかったから，ここで得た結果は記憶域アクセスの相互依存性とは無関係に成立する．

データ重複　多くのデータ配置算法では，1つの記憶装置に蓄えたすべてのデータを次階層の遅い記憶域に重複しておく必要があるから，データの重複の問題が起こる．すなわち，もっとも遅い装置はシステム内の全データのコピーを蓄えなければならないが，そのほとんどのアクセスは速い装置上のデータの複写に対して行われることを意味している．データ重複の影響は，一意的データ貯蔵よりいくらか大きい全システムの記憶容量を必要とし，付加的に速い記憶域を導入すると全体の記憶域サイズが増大することである．性能限界定理を単純な形に展開する際に使った仮定は，容量一定の仮定を含んでいた．データ重複が起こるときは，蓄えられた全体の重複しないデータの和が1になるように，記憶装置のサイズを正規化すれば，(5-3) はなお妥当である．このとき $\sum c_i$ はシステム内の重複の程度に応じて1を越えることになる．データ重複は，所与の投資でより多くの容量を購入しなければならないから，多階層システムの性能を悪化させる．たとえば，例3での数値を使うと，速い記憶域へのアクセスが全アクセスの73パーセントを占めるが，それが全データの10パーセントだけというような二階層記憶域システムでは，同一費用の一階層記憶域システムに対して $\min(T_{avg})=0.5$ を実現できるが，データ重複の場合には同一システムに比較して $\min(T_{avg})=0.59$ となり，一階層記憶域に対して100パーセントの速度改善とはならず，70パーセントの速度改善にとどまる．

7. おわりに

本稿の結論は最適化定理によっている．定理は記憶域の費用対速度特性とアクセス確率とを関係づけている．その解釈が複雑になる場合もあるが，この基本関係は広範囲の記憶域システムに適用できる．この関係は，ここでは機器関数の連続モデルに対して使ったが，離散モデルにおいても同様に有用である．

さらに，その機器関数は，ベキ関数で表現できると仮定した．これは現存する機器仕様を十分に近似する．この仮定を使って，記憶域限界定理を証明した．すなわち最適システムにおいてシステム費用の X パーセントを占める記憶装置は，システムのアクセス遅延の X パーセントの要因となることを示した．均衡基準に合致するシステムは，記憶域のサイズを調節することによって，さらに最適なシステムとなしうる．しかし，費用一定の条件の下ではどのように記憶の速度を調節しても，もはや性能を上げることができない．均衡基準は，広範囲の記憶域階層モデルにして妥当であるが，ベキ関数とならない機器関数に対しては当てはまらない．

機器関数をベキ関数と仮定して解析を進めて性能限界定理を得たが，この定理からは所与の記憶域アクセス確率分布に対して得られる最小平均アクセス時間が得られる．

この限界はある二階層記憶域階層の同一費用の一階層記憶域に対する利点を比べる便利

な手段となる．また，多様な記憶域配置やデータ配置算法を評価する測定技術を確立するのに有用である．

本稿の作成に当たり T. K. M. Agerwala の協力に感謝する．

(1100 ソフトウェア・サポート部 米沢 敬一郎 訳)

参考文献 [1] C. K. Chow, "On Optimization of Storage Hierarchies", *IBM J. Res. Develop.*, Vol. 18, May 1974, pp. 194–203.

[2] C. V. Ramamoorthy and K. M. Chandy, "Optimization of Memory Hierarchies in Multiprogramming Systems," *J. Assoc. Comput. Mach.*, Vol. 17, July 1970, p. 426.

[3] J. E. MacDonald and K. L. Sigworth, "Storage Hierarchy Optimization Procedure," *IBM J. Res. Develop.*, Vol. 19, Mar. 1975, pp. 133–140.

[4] S. L. Rege, "Cost, Performance and Size Trade-offs for Diffierent Levels in a Memory Hierarchy," *Computer*, Vol. 19, Apr. 1976, pp. 43–51.

執筆者紹介 T. A. ウェルチ (Terry A. Welch)

Massachusetts 工科大学で電気工学を専攻．1960, 1962, 1971年にそれぞれ SB, MS, Ph. D. を修得．学位論文は P. Elias 博士の指導で情報検索システムでの検索効率の理論的限界に関するもの．その間，数年 Honeywell 社に所属し，小型コンピュータとの通信の論理設計およびシステム分析に従事．1971，1976年から Austin の Texas 大学助教授で電気工学とコンピュータ・サイエンスを担当．研究分野は記憶システムと記述子に基づくコンピュータの設計．

1976年以降コンピュータ構造部門のマネージャとして Sperry Research Center に所属．仮想記憶システム，VLSI 設計技術，およびコンピュータの信頼性についての研究に参画．

マイクロプロセッサ技術によるプロセッサ設計の考察

Some Considerations in the Design of Mainframe Processors with Microprocessor Technology

G. S. Tjaden
M. Cohn

要 約 メーンフレーム(コンピュータ本体)を多重マイクロプロセッサで実現する場合，主記憶装置に余分な費用がかかるという弊害がある．

ランダム・アクセスのメモリ・チップやマイクロプロセッサという形となった LSI 技術によって，はるかに低価格のコンピュータ・システムを利用することが可能になった．しかし，メーンフレーム・システムには，“マイクロプロセッサ” システムでは不可能ないくつかの特長(たとえば，より高いパフォーマンス，障害に対するより高い許容度)がある．

メーンフレームと同じ機能を LSI 技術で実現できれば，コスト・パフォーマンスが大きく改善できる．そのための設計上の方法として，一組のマイクロプロセッサを相互に結合し，各々のマイクロプロセッサに異なったユーザの仕事をやらせるという方法がよく提案される．この同時並行的に業務処理を行う一組のマイクロプロセッサ全体のスループット(処理能力)は，単一のメーンフレーム・プロセッサのスループットと同じであると考えられている．本稿では，こういう“多重マイクロプロセッサ” の設計アプローチを提案する人々がしばしば見過しているいくつかの考慮すべき点について検討する．

Abstract Implementing mainframes with multimicroprocessors entails an extra cost for main memory. This often-overlooked phenomenon is analyzed here for several mainframe systems.

LSI technology, in the form of random-access memory chips and microprocessors, has led to the availability of computing systems priced very much lower than “mainframe” systems here to fore. Mainframe systems, however, offer some features—higher performance and fault tolerance, for example—that “microprocessor” systems do not.

Achieving mainframe functionality with LSI technology is of great interest to computer designers because of the anticipated dramatic improvement in cost-performance. One design approach that is often proposed is to interconnect a set of microprocessors so that each microprocessor executes different user jobs. The total throughput of this concurrently executing set of microprocessors is expected to be equivalent to the throughput of a single mainframe processor. This paper discusses some considerations which are often overlooked by proposers of this “multimicroprocessor” design approach.

1. はじめに

メーンフレーム・システムのパフォーマンスを表現するために，1秒間に実行される百万単位の命令の数(MIPS)がよく使われる．MIPS の決定には，通常，その主記憶容量と I/O の能力(チャネルや周辺機器の数と種類)が，中央処理装置の処理速度とユーザ・ジョブ上の要求仕様にうまくマッチしているシステム構成で処理されるユーザ・ジョブの典型的な組合せについて，その実行時間を測定することによって決定される．ジョブ・ミックスは，FORTRAN と COBOL のプログラムが大きな割合を占め，ALGOL や PL/I 等のその他の言語で書かれたジョブの占める割合ははるかに少ないのが通例である．これらのジョブは，主に，バッチ処理と会話型処理の両方が混用され，特定バージョンのオペレー

ティング・システムのもとで処理が行われる．実行される機械命令の数とそれらの実行時間から，MIPS単位のパフォーマンスが算出される．

コンピュータ・システムのパフォーマンスを決定するこの方法は，トータル・システムのスループットを基礎としたコンピュータ・システムのコスト・パフォーマンスを比較しようとしているユーザのためのものである．測定結果は，プロセッサや主記憶装置がつくられたときのハードウェア技術だけでなく，その他の数多いパラメタによって左右される．これらのパラメタの中には，オペレーティング・システムの効率，コンパイラのコード変換効率，プロセッサの命令の種類，プロセッサ内のデータ経路の幅，および主記憶装置の語幅があげられる．たとえば，LSI 技術では，マイクロプロセッサのデータ幅と語幅はわずか16ビットに限定されるが，メーンフレーム・プロセッサ用では，ふつう32～36ビットである．

コンピュータ・システムにおける障害の許容度とは，ハードウェアの障害を発見し，データを破壊しないでその障害を修復する能力のことである．過去10年間，メーンフレームにおいて障害の許容度を100パーセントにすることを目指してきたが，まだ達成されていない．ユーザは，障害の許容度が常時向上することを期待するが，このための余分な支出を好まない．現在の多くのメーンフレームは，プログラムの正常な実行中にハードウェアの障害を発見する目的から，プロセッサ内に――少なくとも，データ経路上のパリティや誤り訂正符号用に――余分のハードウェアを有している．ハードウェア障害が発生した場合，その障害の多くが瞬間的なものであるために，実行中の命令を再試行する手段が組み込まれる場合もある．I/O システムでの障害回復のためには，複雑なロールバックと再開始の手続きをオペレーティング・システムの中に組み入れる必要がある．高い可用性が要求されるシステムには，プロセッサ，主記憶，チャネル，周辺機器等により冗長モジュールを含む設計が施されなければならない．また，システムが障害発生と同時にすばやく再構成され，稼動を継続させることができるような手段も含めなければならない．

今日のメーンフレーム・コンピュータには，主記憶システムを実現するために，LSI 技術が広範囲にわたって採用されている．マイクロプロセッサは，デバイス制御装置やインテリジェント端末装置のようなものに取り入れられてきている．一方，使用する論理回路が比較的“ランダムであること”と，高いパフォーマンスと機能性が要求されることから，LSI 技術をメーンフレームのプロセッサ部分に適応するのが，メモリに比べ困難になってきている．この要求を満たすために，現在のマイクロプロセッサ・システムよりもさらに多くのゲートを使って実現する方法に替わってきている．さらに，メーンフレーム・プロセッサを実現するために使われるゲートは，一般的に，マイクロプロセッサで使われるゲート(5～15 ns)よりもはるかに高速のスイッチング速度(0.5～1 ns)を備えている必要がある．

多重マイクロプロセッサについての本稿での分析は，全体のシステム・スループットで測定されるパフォーマンスを中心にしている．障害の許容度というようなメーンフレーム上の機能に関する問題提起は，現在のマイクロプロセッサ・システムがメーンフレームに寄せる期待に追いつけない分野が他にもあることを認識してもらうために提起するにすぎない．

2. 単一プロセッサのメーンフレームを例にして

単一プロセッサのメーンフレームの簡単な例として，図1に示す単純化されたメーンフ

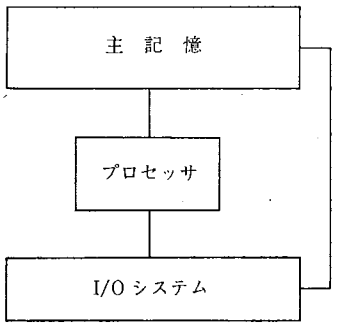


図 1 単一プロセッサからなるメーンフレームのモデル
Fig. 1 Model of a uniprocessor mainframe

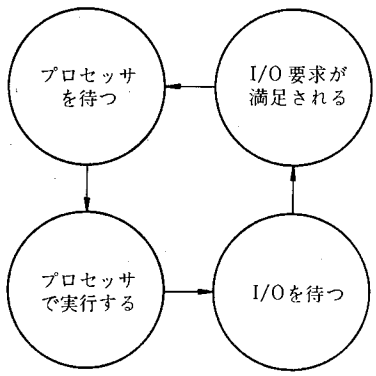


図 2 単一プロセッサからなるメーンフレームのジョブ・ステートの順序
Fig. 2 Job state sequence for a uniprocessor mainframe

レームの単一プロセッサ・システムのモデルを考えてみよう．このシステムは，メーンフレームとして (約 1 MIPS) のパフォーマンスを持つ単一のプロセッサ，オペレーティング・システムとユーザ・プログラム 1 個が常駐できるのに十分な容量を持っている 1 つの主記憶，さらに主記憶に直接アクセスする 1 つの I/O システムとで構成されている．ユーザ・ジョブは，図 2 で示すような“ジョブ・ステート”の順番に従って実行される．この例は，現実的なアプローチとして正確な表現とはいえないが，強調しようとしている主要な点すなわち費用増の問題を浮彫りにするのに役立つものである．

一度に 1 つのジョブを実行するとき，わずか 0.25 MIPS のスループットしか達成しないマイクロプロセッサで図 1 のメーンフレーム・プロセッサ・システムと“同等”のシステムを組み立てるには，おそらく図 3 に示されるように，数個のマイクロプロセッサが必要になるであろう．同等のパフォーマンスを達成するには，少なくとも 4 個のマイクロプロセッサが必要となり，さらに 4 個の主記憶も必要になる．これらの記憶装置は，おのおの図 1 の主記憶と同じ容量のものでなければならない．このように，各マイクロプロセッサの費用がメーンフレーム・プロセッサのわずか 4 分の 1 になるとしても，このマイクロプロセッサ・システムは，3 個の余分な記憶装置を必要とするため，メーンフレーム・システムよりも費用増となる．この例では，より速度が遅く，しかもより安価な記憶装置を

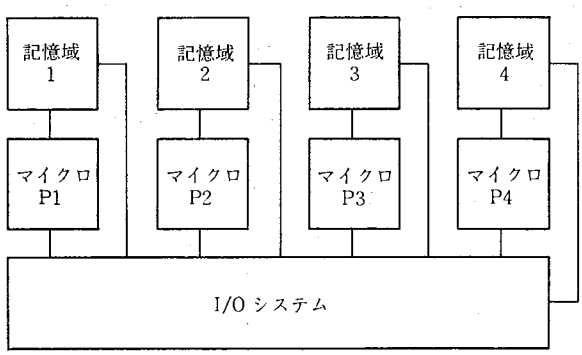


図 3 図 1 の単一プロセッサからなるメーンフレームのモデルと同等である多重マイクロプロセッサ・システム
Fig. 3 A multimicroprocessor system equivalent to the uniprocessor mainframe model in Fig. 1

使用することによって費用を削減することはほとんど不可能である．なぜなら，1 MIPS のプロセッサに使われるメモリよりも，はるかに安いメモリ技術で使用して処理速度が数倍も遅くなるからである．この主記憶の余分な費用はよく見過されるが，無視できない事象である．以下，これらの割増し費用について詳しく分析する．

3. 単一レベル（非仮想）の主記憶を持つメーンフレーム・システム

大部分のメーンフレーム・システムは，きわめて少ない割増し費用でスループットが向上できるため，“多重プログラム化” されている．すなわち，1つのユーザ・プログラムはそれがI/O要求を出すまで実行され，その要求が出た時点で，次の別のユーザ・プログラムに移る．新しいユーザ・プログラムはそれがI/O要求を出すまで実行される．I/O要求が満たされたユーザ・プログラムは，それ以降の実行のために，オペレーティング・システムによって待ち行列に入れられる．プロセッサとI/Oシステムはともに同時並行で実行しているために，両方とも主記憶をアクセスしなければならない．したがって，主記憶の帯域幅（bandwidth）が，プロセッサのパフォーマンスあるいはI/Oシステムのパフォーマンスのどちらかを劣化させるほど，小さいことは許されない．

この種の多重プログラム化された単一プロセッサ・システムのパフォーマンスは，S. E. Madnick と J. J. Donovan[1] によってモデル化され分析されている．そのモデルの重要なパラメタは，主記憶の中に常駐化されなければならないユーザ・ジョブの数（多重プログラミング度と呼ばれ，n で表す）である．彼らは Markov の行列モデルを使い，多重プログラミング度の関数として，プロセッサの利用度と個々のジョブについての “I/O 待ち比率” (ω) を計算した．I/O 待ち比率とは，もしユーザ・ジョブがシステム上で走っている唯一のジョブ（多重プログラミング度が1）であるなら，そのユーザ・ジョブのI/O要求が出されてから満たされるまでの全時間の分数である．図4は，S. E. Madnick と J. J. Donovan のモデルについての典型的な結果を示すものである．したがって，もし単一ユーザ・ジョブの場合で ω が65パーセントであるなら，プロセッサはその時間の65パーセントが遊んでいることになる．たとえば，2個のユーザ・ジョブがシステムに与えられていれば，最

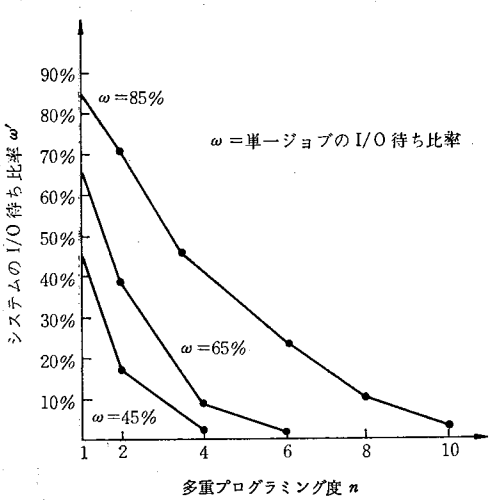


図 4 単一プロセッサ・システムの分析
Fig. 4 Uniprocessor system analysis

初のジョブがI/O待ちの間，次のジョブはプロセッサを使い続けることが可能である．このモデルに従えば，こういう状況は結果として全プロセッサの40パーセントが遊んでいることになる(これをシステムのI/O待ち比率と呼び，ω'で表す)．このモデルでは，すべてのジョブに同一のI/O待ち比率が与えられ，しかも使用される多重プログラミング度と同数かそれ以上のシステムのチャネル数があるという条件を仮定している．

このモデル化のアプローチに見通しをつけるため，ある特定の典型的な単一プロセッサ・システムを次に調べる．M. M. Lehman[2]によると，IBM 360/65 や 370/155 の代表的なシステムでは，ユーザ・ジョブの平均的サイズは128キロバイトで，各々のジョブの平均I/O待ち比率であるωは，65パーセントであるという．このモデルからすると，システムのI/O待ち比率を1パーセント以下にするためには，多重プログラミング度としてユーザ・ジョブを6個にしなければならない．そうすると図2の主記憶は，オペレーティング・システムや分割効果のための予約領域の他に，各々が128キロバイトのユーザ・ジョブ6個分(768キロバイト)を収容するのに十分な容量でなければならない．

図1のシステムと同等の多重マイクロプロセッサは，N個で構成される一組のパフォーマンスの低い(LSIマイクロプロセッサとして実現された)プロセッサで置き換えられる．これらN個の各プロセッサは，図1のプロセッサと同じ多重プログラム形式でユーザ・ジョブをエミュレートしたり，実行したりする．このシステムを図5で示す．わずか1つのオペレーティング・システムしか必要としない点で，図3の方法よりも，主記憶をマイクロプロセッサ間で共有する図5の方法がはるかに有利である．このシステムは，そのためのオペレーティング・システムがすでに存在している古典的な多重プロセッシング・システムである．

このアプローチの分析では，N個のマイクロプロセッサによる主記憶と，オペレーティング・システムのコンテンションとに起因するパフォーマンスの劣化について無視することにする．これは主記憶の大きさに及ぼす影響を強調するためである．図5とは異なったI/Oシステムの相互結合の配置も可能であるが，結合方法の相異は結果的にはたいした影響がないため，本稿では考慮しない．

S. E. Madnick, J. J. Donovan のモデルは，この種の多重マイクロプロセッサの構成には直接適用できない．なぜなら，数個のプロセッサが1つの主記憶内に蓄えられた数個のユーザ・ジョブのために実行するというケースをモデル化していないからである．この構成を取り扱う Markov チェーン・モデルは次のような仮定に基づいて開発された．

1) 主記憶には，マイクロプロセッサの数と少なくとも同数のジョブが常駐している．
2) 主記憶には，チャネルの数と少なくとも同数のジョブが常駐している．

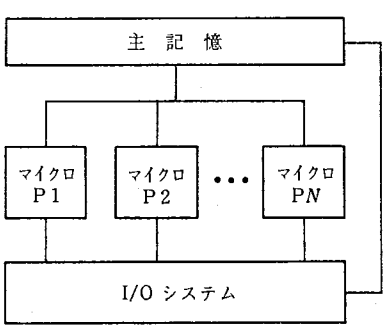


図 5　図1のプロセッサのみを置換できる同等の多重マイクロプロセッサ
Fig. 5　Multimicroprocessor equivalent replacing only the processor of Fig. 1

このモデルの開発には，行列理論がそのまま使用される．また，マイクロプロセッサの個数 N，チャネル数 C，さらにユーザ・ジョブの数 n の関数として，CPU の遊び時間が算出される．本稿で紹介する特定のシステム分析はこの新しいモデルを基礎にしている．

多重マイクロプロセッサで実行されるユーザ・ジョブは，それぞれ単一プロセッサ・システムで実行される場合と同じ I/O システムのサービス時間を必要とする．しかし，これらのジョブの I/O 待ち比率は，CPU 時間が単一プロセッサの場合よりも長いため，同一ではない．公平に比較するために，多重マイクロプロセッサ・システムの“有効な”システム・スループットが単一プロセッサ・システムと同じでなければならないということに注意しなければならない．この同等性を達成するための 1 つの方法は，各マイクロプロセッサの多重プログラミング度を調整(主記憶上のジョブを増減)して，それぞれのジョブに単一プロセッサの場合と同じ I/O 待ち比率を達成させることである．もし，MIPS での単一プロセッサの能力を P とすれば，N個のマイクロプロセッサが必要とする能力は P/N とになる．第 2 の方法としては，マイクロプロセッサをより高い I/O 待ち比率(より低いプロセッサの利用度)で稼動させる一方，各マイクロプロセッサの低いスループットを穴埋めするために，マイクロプロセッサの個数を増すというやり方である．同様に，低いマイクロプロセッサの I/O 待ち比率をとれば，結果としてマイクロプロセッサの個数を少なくする必要がある．最高のコスト・パフォーマンスを得る方法は，記憶装置とマイクロプロセッサの相対費用に左右される．新しく開発されたモデルによって，このトレード・オフが特殊なケースで可能になっている．結論を変更することなく論点を単純化するために，単一プロセッサの I/O 待ち比率と等しいマイクロプロセッサの I/O 待ち比率を使う 1 番目の方法を本稿では仮定する．

多重マイクロプロセッサが，期待どおりの有効なシステムの I/O 待ち比率を得るために維持されなければならない多重プログラミング度は，パラメタ N (マイクロプロセッサの個数)，C (チャネルの数)，ω によって求める．すなわち，N と C と ω のいろいろ異なっ

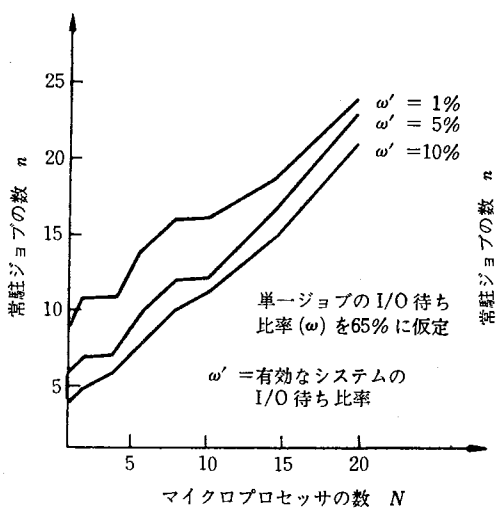


図 6 システムの I/O 待ちの拘束を満足するために必要とされる常駐ジョブの数

Fig. 6 Number of resident jobs required to meet system I/O wait constraints

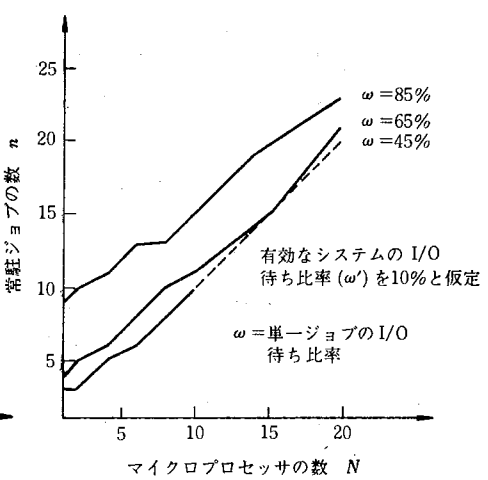


図 7 単一ジョブの I/O 待ち比率に関する各種の数値に必要な常駐ジョブの数

Fig. 7 Number of resident jobs required for various values of single-job I/O wait percentage

た数値でモデル式を解き，単一プロセッサのI/O待ち比率を同等のマイクロプロセッサのI/O待ち比率に合わせることによって決定される．

得られた結果をグラフ化した図6と図7は，単一ジョブのI/O待ち比率 ω と有効なシステムのI/O待ち比率 ω' とのいろいろな異なった仮定値の曲線群を示すものである．チャネル数を指定するパラメタの曲線群が表示されていないのは，チャネル数を仮定してもチャネル容量が飽和状態でないかぎり，多重プログラミング度にはたいして影響を与えないことが明らかになったためである．

すべての場合において，多重マイクロプロセッサの構成では，ある一定レベルのシステムのI/O待ち(プロセッサ利用度)を達成するためには，単一プロセッサの構成時よりも多くの常駐ジョブが必要となる点に留意すべきである．たとえば，10個のマイクロプロセッサからなるシステムは，単一プロセッサ・システムよりも6個ないし7個も多いジョブを主記憶に常駐させておかなければならない．また，20個のマイクロプロセッサからなるシステムでは，主記憶に約13個もの余分なユーザ・ジョブを常駐させておく必要がある．おもしろいことに，このモデルは，B. R. Borgerson[3] の単純な線形モデルが予言した(マイクロプロセッサが1個増えるたびに1個の常駐ジョブを追加する必要があるという)よりも，さらに少ない常駐ジョブを追加すればよいということを示している．

ある一定数のマイクロプロセッサに対して，ある一定レベルのシステムのI/O待ちを達成するために必要な余分な常駐ジョブの個数が，すべてのユーザ・ジョブを対象とした単一ジョブのI/O待ち比率とは，あまり関係しないことを，図7は示している．単一ジョブのI/O待ち比率 ω が45パーセントを示す点線は，実数というよりはむしろ必要とされる常駐ジョブの上限である．実数は，このモデルが妥当な領域の外にあるため決定することが不可能である．20個のマイクロプロセッサからなるシステムに対する割増し記憶は ω が85パーセントの場合，14個のユーザ・ジョブを収容するのにふさわしい容量でなければならない．また，ω が65パーセントの場合の割増し記憶は，17個のユーザ・プログラムを収容しうるものでなければならない．

図6と図7で示された結果は，ある仮定のシステムについての費用分析を行うことによって，一層よく評価することができる．現存するプロセッサの廉価版を20,000ドルの再生産費用で製作するために，多重マイクロプロセッサのアプローチを使うのが望ましい．システム費用の目標を達成するには，個々のマイクロプロセッサの再生産費用はいくらであるべきなのであろうか．

「コンピュータ・ハードウェアの費用傾向」という D. A. Hodges の調査[4]によると，OEM市場に対する1979年の記憶システムの価格は，ビット当たり約5セント(1981年では約3セント)になっている．分割効果を無視すれば，128キロバイト（4バイトごとに6ビットのECCを含める）の典型的なユーザ・ジョブ1個を記憶するための主記憶は約600ドルになる．したがって10個のマイクロプロセッサからなるシステム ($n=10$) では，$\omega=65\%$，$\omega'=5\%$ と仮定して，多重プロセスによる主記憶費用の全増加額は3,600ドルとなる．この記憶費用の増加分は，プロセッサ費用の一部として考えなければならない．それゆえ，各個別のマイクロプロセッサは，多くて (20,000ドル−3,600ドル)/10=1,640 ドルの再生産費用が与えられることになる．このとき，多重アクセスの記憶とI/Oシステムに対する要求に起因する追加費用のオーバヘッドは無視している．20個のマイクロプロセッサからなるシステムでは，10,200ドルの費用で17個の割増し常駐ジョブが必要となり，結局1個のマイクロプロセッサ当たりの最大許容費用は490ドルになる．マイクロプロセッ

サの個数がそれ以上になる場合には，プロセッサの全費用は割増し主記憶に対する代価を払って割り振られる．

この例の場合，マイクロプロセッサが10～20個の間で構成されるシステムを設計することを正当化するのは非常にむずかしいであろう．$n=20$ の場合，各マイクロプロセッサは，わずか490ドルしかかからないのに対し，$n=10$ の場合は，各マイクロプロセッサに1,640ドルもの費用が必要になる．したがって，20個のマイクロプロセッサからなるシステムを10個のマイクロプロセッサのシステムに変更するには，コスト・パフォーマンスの面から見た場合，1個のマイクロプロセッサ当たりの費用が3倍以下の増加となるものの，各マイクロプロセッサのパフォーマンスを2倍にするだけでよいことになる．このようなトレード・オフを有利に実現できないような状況を想像することは困難である．マイクロプロセッサが5個以下で構成されているシステムの場合でも，必要とされるパフォーマンスを2倍化するには，2倍以上の許容費用の増加分を伴う．このような古典的な多重プロセッサのアプローチを使って，多重マイクロプロセッサを用いたメーンフレーム・システムを設計する場合，経済的な意味からプロセッサ・アーキテクチャが単一マイクロプロセッサのパフォーマンスを最大限にしなければならなくなるのは明らかである．

4. 仮想主記憶を持つメーンフレーム・システム

仮想記憶のシステムでは，ユーザ・ジョブのワーキング・セットだけが主記憶に常駐する．一般にワーキング・セットのサイズは，全プログラム・サイズの約半分であることが判明している[1]．したがって，128キロバイトの通常プログラムでは，わずか64キロバイトの主記憶が供給されればよい．多重マイクロプロセッサのアプローチでは，この主記憶サイズの削減が主記憶のオーバヘッドに関する問題を解決すると，一般の人達は期待するかもしれない．この章では，この期待がどの程度まで実現できるかについて分析する．

ユーザ・ジョブごとに4キロバイトのページと64キロバイトのページが割り当てられているページ・オン・デマンドのシステムを例としてとりあげる．補助記憶として，平均アクセス・タイムが5ミリ秒のドラムを仮定する．したがって，1つのページ障害 (page fault) によって，I/O 要求の場合と同様に，プログラム実行で5ミリ秒の遅延が生ずる．“典型的な”ページ障害率は不明である．本稿で引用するページ障害率は，プログラム・サイズの半分に等しいワーキング・セットのサイズに対して 10^6 の実行命令当たり185のページ障害率，すなわち1命令当たり 1.85×10^{-4} の障害が発生するプログラム例[5]からとることにする．われわれが仮定した通常のワーキング・セット(プログラム・サイズの50パーセント)に対して測定したページ障害率が，実行された1命令当たり約 1×10^{-3} と高率[6,7]であったため，この障害率はわれわれの目的にとってきわめて控え目な数字であるように思われる．

プロセッサが1MIPS，ページ障害率が 1.85×10^{-4}，1ページ当たりのページ・スワップ遅延を5ミリ秒とすれば，百分率で表したページ待ち時間 $(T_{ページ})/(T_{ページ}+T_{CUP})$ は50パーセントである．次に，I/O 待ち比率が65パーセントで，通常の I/O サービス時間が30ミリ秒であるわれわれの典型的なプログラムでは，全有効待ちの比率である $(T_{ページ}+T_{I/O})/(T_{ページ}+T_{I/O}+T_{CUP})$ は70パーセントになる．図7の $\omega=65\%$ と $\omega=85\%$ との曲線から補外すれば，仮想記憶を持つ単一プロセッサ・システムでは非仮想主記憶のシステムよりも，もう1つ多い常駐ジョブが必要になることが判明する．しかしながら，プログラム・サイズの半分であるワーキング・セットの全サイズは，約40パーセント小さくな

る．10個のマイクロプロセッサで構成されるシステムではワーキング・セット6個の記憶サイズの増分が必要になるであろう．一方，20個のマイクロプロセッサで構成される仮想記憶システムでは，14個の割増しワーキング・セットが常駐していなければならない．

前にあげた簡単な例を引用すれば，ワーキング・セット1個(64キロバイト)の記憶費用は300ドルである．したがって，1台20,000ドルの単一プロセッサを1台の多重マイクロプロセッサで代替するには，10個のマイクロプロセッサで構成されるシステムの場合，1マイクロプロセッサ当たりの最大費用が1,820ドルになる．また，20個のマイクロプロセッサで構成されるシステムでは，1マイクロプロセッサ当たり790ドルになる．この比率は2.3：1である．これに相応する非仮想記憶のシステムが3.35：1であることと比べると，相当大きい改善である．10～20個までのマイクロプロセッサで構成される仮想記憶システムの多重マイクロプロセッサの生存性について決定をくだすには，次のようなオーバヘッド費用を詳細に分析してからでなければならない．

1) 1マイクロプロセッサ当たりのページ・マップ用ハードウェア
2) オペレーティング・システム上の付加的な複雑性
3) オペレーティング・システムを実行する場合の付加的なタイム・オーバヘッド

5. 多重マイクロプロセッサを採用したシステムでの時分割処理

時分割処理(タイム・シェアリング)のために，多重マイクロプロセッサによるアプローチ方法を使う概念に対しては，ある種の直観に訴えるものがある．それは，低価格で低パフォーマンスのマイクロプロセッサでも高レベルの応答性が維持されると考えられるごく少数の使用ユーザに対して各マイクロプロセッサが割り当てられる，という推測である．この章では，このようなアプローチを分析し，さらに古典的な単一プロセッサによる時分割システムとの対比を行うことにする．このとき，もう一度主記憶の費用を比較の基準として用いる．

単一システムとして，仮想記憶を持つ1MIPSのプロセッサを仮定する．プロセッサの時間はタイム・スライスで管理され，記憶域の管理は“デマンドによってロードし，ポスト・ページを行う”方式[1]のもとで行われる．この方式では，ジョブがそのタイム・スライスの終わりに主記憶域から追い出され，ふたたびプロセッサに割り当てられると，ページ・オン・デマンドで再度ロードされる．多重プログラミングをもう一度採用する．ふつう主記憶に常駐しているジョブ数よりも，さらに多くのユーザがシステムを使用する．ジョブが与えられた時間の後に追い出されるのは，もちろんこの理由からである．前に仮定した典型的なプログラム特性をこの分析でも使うことにする．仮想記憶の特性は前出の例と同じであると考える．

スワッピングとタイム・スライシングによって，各ユーザ・ジョブごとに，付加的な待ちオーバヘッドが生ずる．そのため，非時分割の状態と比べて有効待ち比率が高くなる．転送率が4×10^6バイト/秒の補助記憶と，前例のように平均アクセス時間がミリ秒の補助記憶を仮定すれば，典型的な64キロバイトのワーキング・セットを主記憶域からアンロードするための全時間は，約5(ミリ秒)＋16(ミリ秒)＝21ミリ秒になる．待ち時間の計算は，多少複雑であるが，これは，ページ・オン・デマンドでワーキング・セットが再度ロードされるためである．測定結果では，ワーキング・セットが設定されている最中のページ障害率の方が，全体の平均ページ障害率よりもはるかに高くなっている．われわれのケースの場合，16ページ分のワーキング・セットが設定されている間の1実行命令当たりの障害

は，1.85×10^{-3} という初期ページ障害率を得ることになる．普通のページ障害率では，ワーキング・セットの設定時間中にわずか2回くらいの障害しか起きないことになり，ワーキング・セット設定のためのI/O転送の正味回数は14になる．したがって，各タイム・スライスごとのワーキング・セット設定に要する正味待ち時間は，14(障害)×6(ミリ秒/障害)＝84ミリ秒である．そのため，1タイム・スライス当たりの総追加待ち時間は105ミリ秒になる．

結果として，有効待ち比率はタイム・スライスの大きさの関数である．われわれは，非常に応答性の高い会話型の時分割システムを供給したいため，タイム・スライスの値が50から100ミリ秒の範囲内のものに興味がある．タイム・スライスの値が50および100ミリ秒の場合，いままで述べてきた典型的なユーザ・ジョブでの有効待ち比率はそれぞれ82パーセント，77パーセントである．したがって時分割システムでは，常駐ワーキング・セットの総数が非時分割システムの場合よりも多くなる．

6. おわりに

本稿は，多重マイクロプロセッサのアプローチに関して，費用面の主要なトレード・オフの1つを分析する技術を紹介するものであって，このアプローチが意味ある諸環境を全体的に分析したものではないことを強調したい．20,000ドルという費用拘束は，SSI と MSI で実現された1979年の妥当な 1 MIPS プロセッサと思われるものに対する1つの評価にすぎない．費用拘束が大きくなればなるほど，明らかに主記憶の費用オーバヘッドは目立たなくなるであろう．しかしながら，オーバヘッドは依然として存在するであろうし，システム設計中，忘れてはならないものである．以上のような考え方に基づいて，B. R. Borgerson, G. S. Tjaden, M. L. Hanson の3人[8]は，メーンフレーム・プロセッサを実現するために LSI を使い，しかもこの多重マイクロプロセッサのアプローチが持っている不都合を有していない異なった設計方法を提案した．記憶スイッチ，コンテンションによるパフォーマンスの劣化，オペレーティング・システムの増大する複雑性等，マイクロプロセッサに関するその他のオーバヘッドを含めれば，このアプローチの有効性はさらに減少するであろう．

最後に，これらの結果は，今日の典型的なユーザ・ジョブにのみあてはまるということを指摘しておきたい．新しいソフトウェア構成によって，多重マイクロプロセッサ・システムで，より生存性の高いホストが生まれてくるかもしれない．しかし，少なくとも G. M.Amdahl[5]は，そのことに対し疑問を抱いている．本稿の結果から，多重マイクロプロセッサに関する主記憶のオーバヘッド問題を解決するためには，単一のジョブが同時並行的にマイクロプロセッサで実行できるほどの多数の小さいタスクに分けられるような新しいソフトウェア構成にしなければならないと推測されよう．

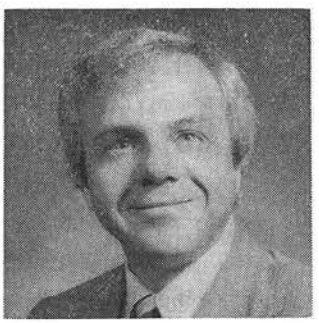
なお，主記憶オーバヘッド問題を論ずるよう勧めてくれたことに対し B. R. Borgerson, G. A. Champine そして M. L. Hanson に謝意を表する．

(技術情報管理部 熊谷 貴 訳)

参考文献 [1] S. E. Madnick and J. J. Donovan, *Operating Systems*, McGraw-Hill, 1974.
[2] M. M. Lehman and J. J. Rosenfeld, "Performance of a Simulated Multiprogramming System," *AFIPS Conf. Proc.*, Vol. 33, 1968 FJCC, pp. 1431-1442.
[3] B. R. Borgerson, "The Viability of Multimicroprocessor Systems," *Computer*, Vol. 9, No. 1, Jan. 1976, pp. 26-30.

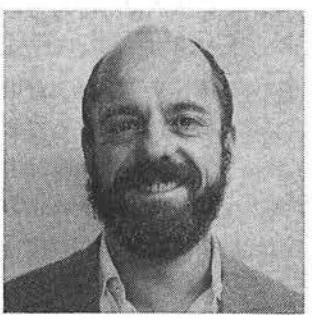
[4] D. A. Hodges, "Trends in Computer Hardware Technology," *Computer Design*, Vol. 15, No. 2, Feb. 1976, pp. 77–86.
[5] G. M. Amdahl, "Validity of the Single Processor Approach to Achieving Large Scale Computing Capabilities," *AFIPS Conf. Proc.*, Vol. 30, 1967 SJCC, pp. 483–485.
[6] E. G. Coffman and L. C. Varian, "Further Experimental Data on the Behavior of Programs in a Paging Environment," *Comm. ACM*, Vol. 11, No. 7, July 1968, pp. 471–474.
[7] W. W. Chu and H. Opderback, "Performance of Replacement Algorithms with Different Page Sizes," *Computer*, Vol. 7, No. 11, Nov. 1974, pp. 14–21.
[8] B. R. Borgerson, G. S. Tjaden and M. L. Hanson, "Mainframe Implementation with Off-the-Shelf LSI Modules," *Computer*, Vol. 11, No. 7, July 1978, pp. 42–45.

執筆者紹介 G. S. チャデン (Garold S. Tjaden)



Sperry Univac 社のメジャ・システムズ・ディヴィジョンのアドバンスト・テクノロジ部長，将来のメーンフレーム・システムとそれに必要とされる基本的テクノロジに関する研究計画に従事．また同社の超 SLI 開発について全体制を計画・設計する責任を持つ．過去 Sperry Research Center でマイクロプロセッサのビルディング・ブロック方式をメーンフレーム・システムの設計に適応する研究に従事．
コンピュータ・サイエンスの Ph. D を Jones Hopkins 大学から，MSEE を Northwestern 大学から，さらに BSEE を Utah 州立大学から取得．IEEE のコンピュータ・ソサィアティ部会と ACM の会員．

M. コーン (Martin Cohn)



1961年から Sperry Research Center に勤務，離散系数学担当マネージャであり，主としてチャネル・マッチング・コード，データコンプレッション・コード，ディジタル・シークェンスの設計に従事．
Harvard 大学と Boston 大学の時間講師でもあり，数学の AB，応用数学の AM と Ph. D をそれぞれ 1956年，1957年，1961年に Harvard 大学から取得．

新しい差分法の提案と数値実験

An Attempt to Flexible Finite Difference Method and Its Applications

藤野 勉
渡部義維

要 約 有限要素法が開発されるまでは，等間隔直交格子系差分法による微分方程式の数値解法が連続体解析に利用されてきた．この方法は，手続きが簡単で直接的であること，計算精度が良好であること，等の長所を持つが，複雑な幾何学的形状を持つ境界に適合しがたい欠点もある．ここでは，次に述べるような方法により，境界適合性を改善した差分法についての研究結果を述べる．

1) 直交，等間隔格子系において格子間隔を必要に応じ，1/2, 1/4, 1/8, ･･･ と短縮し，境界適合性を検討する．
2) 境界形が格子内で直線に近くなるまで格子を短縮し，最終的には直交，不等間隔差分を行う．
3) 境界の形に即した直交曲線座標格子系またはそれに近い格子系がつくられれば，その格子点を節点とする差分式を求める．
4) 直交曲線座標格子系が求められない場合，任意に分布された節点について最小2乗法により最適差分式をつくる．
5) 任意分割の有限要素法と任意節点配置の差分法を比較するとき，一般に精度では後者が優れ，係数行列の対称性では前者が優れている．たとえば，Laplace 方程式を解くとき，節点1次A三角形要素系による有限要素法と6節点2次展開の差分法とを比較する．前者では2次精度は満たされず，対称性は保持されるが，後者では2次精度は保証されるが対称性は損われる．

上の計算法を，いくつかの数値実験によりその有効性を確かめた．

Abstract A flexible finite difference method in orthogonal curvilinear nets is proposed to yield simple computational procedures and a superior fitness into the complicated boundaries for the numerical solution of partial differential equations.

The results of a numerical experiment are also supplemented to prove its wide practical applicability and desired accuracy by the use of a certain potential flow model.

1. はじめに

連続体解析には等間隔直交格子系差分法および有限要素法が用いられているが，いずれの方法にも一長一短があり，その優劣は容易に論じられない．本稿では，その両者の優れた点を合わせ，直交曲線座標格子系による差分方程式法について論じる．

まず以下に，有限要素法と差分法の優劣について簡単な考察を加える．

1) 有限要素法は基本的にはエネルギ領域の計算である．そのため与えられた微分方程式，および境界条件式について，それが完全変分（自己随伴形微分方程式）のときは汎関数を求め，不完全変分（非自己随伴形微分方程式）のときは重み関数をかけ，Green の

第1積分によりエネルギ関数の積分形に変形する．次に要素分割を行い，要素ごとに積分を実行し，これを全系に加算，さらに変数ごとに再整理して最終的な離散式に達する．これに対し差分方程式法では，微分方程式より直接差分方程式に変換される．

2) 差分法では，その計算精度を良好な状態に置くために，系が等間隔直交格子に近いことが必要である．そのため直交曲線座標格子系を使用することが有効である．任意の形の境界で囲まれた領域内に正確な直交曲線座標格子系をつくることは，それ自身一連の計算作業を必要とする．このことから，差分法が一見不利なように思われる．しかし実際には，厳密な意味の直交曲線座標格子系は不必要で，フリーハンドによる近似的な曲線格子系で十分である．

3) 一般に $2n$ 次精度の計算を行うとき，有限要素法では n 次要素，差分法では $2n$ 次の家族構成が必要である．たとえば2次精度の計算を行うとき有限要素法では1次要素の形状関数を求めるための3×3行列について，差分法では6×6の行列についてその逆行列を求める必要がある．同様に4次精度の場合，有限要素法では6×6の行列について，差分法では15×15の行列について，その逆行列を求める．ここで家族とは，差分評価格子点を長とし，長に情報を提供する格子点の集りである．2次精度ならば6個の格子点で，4次精度ならば15個の格子点で，その家族は構成される．

4) かなり大きな分割について，有限要素法では家族数概略 n^2，差分法では $2n^2$ である．n は領域を n 方形とするときの分割数である．

以下，直交格子系で格子間隔を 1/2, 1/4, ･･･ と短縮して境界適合性を改善した計算法，直交曲線座標格子系ならびに最小2乗法を用いた任意節点配置による差分法を提案し，さらに直交曲線座標格子系について行ったいくつかの数値実験について報告する．

2. 任意節点配置による差分方程式の導入

等間隔直交格子系により導入された差分方程式では，その精度は良好であるが，複雑な幾何学的形状を持つ境界への適合性は劣っている．この点を改善するため任意節点配置による差分化を試みる．微分方程式をある与えられた点で差分化するとき，その節点配置の選び方がもっとも重要な問題となる．いま簡単にするため2次元問題を取り扱うこととし，任意の点 $P_i(x_i, y_i)$ の近傍における変数 $u(x, y)$ の挙動を Taylor 展開により次に示す．

$$u(x,y)=u(x_i,y_i)+(x-x_i)u_x(x_i,y_i)+(y-y_i)u_y(x_i,y_i)$$
$$+\frac{1}{2}(x-x_i)^2u_{xx}(x_i,y_i)+(x-x_i)(y-y_i)u_{xy}(x_i,y_i)$$
$$+\frac{1}{2}(y-y_i)^2u_{yy}(x_i,y_i)+\cdots \qquad (2\text{-}1)$$

ただし，$u_x=\partial u/\partial x$, $u_y=\partial u/\partial y$, $u_{xx}=\partial^2u/\partial x^2$, $u_{xy}=\partial^2u/\partial xy$, $u_{yy}=\partial^2u/\partial y^2$ である．さらに，簡単にするために P_i 原点を移した座標 $x-x_i, y-y_i$ を便宜上 x, y で表すこととする．一般に(2-1)を

$$u(x,y)=\sum_1^N\alpha_nf_n(x,y) \qquad (2\text{-}2)$$

とするとき，2次展開ならば $N=6$ として

$$f_1=1,\ f_2=x,\ f_3=y,\ f_4=x^2/2,\ f_5=xy,\ f_6=y^2/2 \qquad (2\text{-}3)$$
$$\alpha_1=u,\ \alpha_2=u_x,\ \alpha_3=u_y,\ \alpha_4=u_{xx},\ \alpha_5=u_{xy},\ \alpha_6=u_{yy} \qquad (2\text{-}4)$$

3次展開ならば $N=10$ として，さらに

$$f_7=x^3/6,\quad f_8=x^2y/2,\quad f_9=xy^2/2,\quad f_{10}=y^3/6 \tag{2-5}$$

$$\alpha_7=u_{xxx},\quad \alpha_8=u_{xxy},\quad \alpha_9=u_{xyy},\quad \alpha_{10}=u_{yyy} \tag{2-6}$$

を加える．4次展開ならば $N=15$ として，以下同じように行う．

2次精度の展開において，原点以外の節点 P_m $(m=2\sim6)$ とするとき，P_1 を長とする節点の集りを1つの家族を構成する単位と考える．いま $f_{mn}=f_n(x_m, y_m)$ によって行列 f_{mn} を定義するとき，次の関係がある．

$$u_m\equiv u(x_m, y_m)=f_{mn}\alpha_n \tag{2-7}$$

ここでは n について総和規約が用いられている．(2-7)を α_n について解き，

$$\alpha_m=f^{-1}_{mn}u_n \tag{2-8}$$

をつくる．このように α_m が求まるためには f_{mn} の逆行列 f^{-1}_{mn} が存在すること，すなわち f_{mn} が正則であることが必要である．一般に任意の節点配置に対して f_{mn} が正則か否か，さらに意味を拡張して，性質のよい節点配置か否かを判断することは非常に困難である．ところが，等間隔直交格子系では，その格子点を節点とする家族が正則な集りであるか否かが比較的容易に判定できる．このように等間隔直交格子系によって得られた知識を任意節点配置系に拡大するためには，節点を直交曲線座標格子系の格子点に定めればよい．しかも実際には近似的な直交曲線座標格子系で十分であるため，実用性があると思われる．

3. 等間隔直交格子系における格子点配置

2次，3次，4次精度の展開系について正則な配置の考察を行う．

3.1 2次精度展開格子系

2次精度の展開を行うため，6個の格子点を不用意に選定すると，目的に添わない家族構成となる．このことは，次の例によって示すこととする．

例 1　格子間隔 h の等間隔直交格子系で，格子点がすべて2本の直線上にのっている図1(a)のような配置系を考える．この配置は正則でない．事実，格子点間に $3u_1-u_2-3u_3+u_4=0$ の線形結合関係が存在し，u_{xy} を求めることはできない．

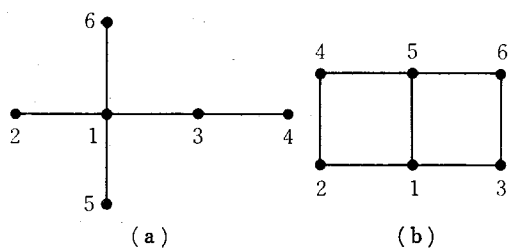


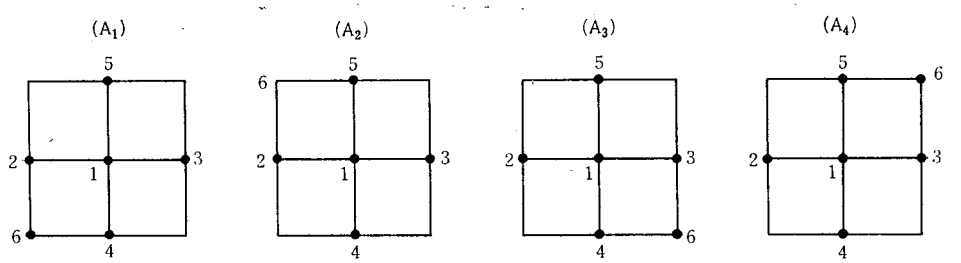
図 1　格子点配置系
Fig. 1　Sample irregular difference families

例 2　図1(b)に示す格子点配置でも格子点は2本の直線上にあるので正則な配置ではない．この配置では $2u_1-u_2-u_3+u_4-2u_5+u_6=0$ の線形結合関係が存在し，u_y, u_{yy} を定めることはできない．一般に正則であるためには $\det f_{mn}\neq0$ が満たされていることが必要である．格子点がすべて1つの2次曲線(2本の直線の集りを含む)上にあるならば上の条件は満たされなくなり，その配置は正則ではなくなる．

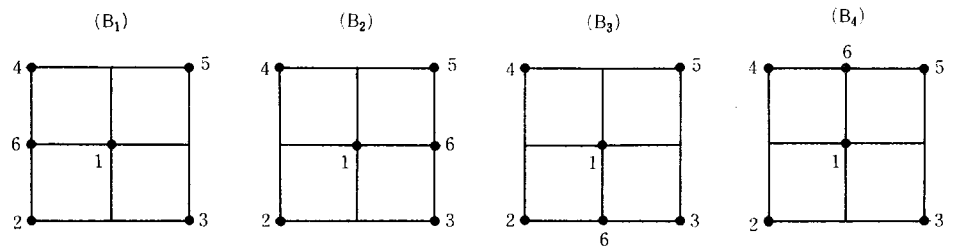
次に正則な格子点配置について代表的なものを述べる．

例 3　十字形格子点配置(図2(a))　家長格子点 P_1 を中心として，他の格子点をその上下左右ならびに対角上に1点配置したものである．この配置では $P_1\sim P_2$ を通過する2本の直線 $x=0, y=0$ が決定され，P_6 はその直線外に位置している．上の配置により，

$$u_x=(-u_2+u_3)/2h \qquad u_y=(-u_4+u_5)/2h$$
$$u_{xx}=(-2u_1+u_2+u_3)/h^2 \qquad u_{yy}=(-2u_2+u_4+u_5)/h^2$$



(a) 十字形格子点配置



(b) X字形格子点配置

図 2 格子点配置

Fig. 2 Difference families of cross shape and rectangular shape

$$u_{xy}=\begin{cases}(u_1-u_2-u_4+u_6)/h^2 & (A_1),\quad (-u_1+u_2+u_5-u_6)/h^2 & (A_2)\\ (-u_1+u_3+u_4-u_6)/h^2 & (A_3),\quad (u_1-u_3-u_5+u_6)/h^2 & (A_4)\end{cases}\qquad (3\text{-}1)$$

が求められる．たとえば Laplace 方程式は

$$u_{xx}+u_{yy}=(-4u_1+u_2+u_3+u_4+u_5)/h^2 \qquad (3\text{-}2)$$

例 4 X字形格子点配置(図 2(b)) 家長格子点 P_1 を中心として対角格子点に 4 点と，上下左右いずれかの格子点に 1 点配置する．この配置では P_1〜P_5 は P_1 を通過するX線上にのり，P_6 はそれら直線外に位置しているので正則である．微係数の差分表示は，

$$\left.\begin{aligned}&u_x=(-u_2+u_3-u_4+u_5)/4h \qquad u_y=(-u_2-u_3+u_4+u_5)/4h\\ &u_{xy}=(u_2-u_3-u_4+u_5)/4h^2\\ &u_{xx}=\begin{cases}(-4u_1-n_2+u_3-u_4+u_5+4u_6)/2h^2 & (B_1)\\ (-4u_1+u_2-u_3+u_4-u_5+4u_6)/2h^2 & (B_2)\\ (u_2+u_3-2u_6)/h^2 & (B_3)\\ (u_4+u_5-2u_6)/h^2 & (B_4)\end{cases}\\ &u_{yy}=\begin{cases}(u_2+u_4-2u_6)/h^2 & (B_1)\\ (u_3+u_5-2u_6)/h^2 & (B_2)\\ (-4u_1-u_2-u_3+u_4+u_5+4u_6)/4h^2 & (B_3)\\ (-4u_1+u_2+u_3-u_4-u_5+4u_6)/4h^2 & (B_4)\end{cases}\end{aligned}\right\}\qquad (3\text{-}3)$$

である．Laplace 方程式は，次のように表される．

$$u_{xx}+u_{yy}=(-4u_1+u_2+u_3+u_4+u_5)/2h^2 \qquad (3\text{-}4)$$

ここで格子間隔の短縮について述べる．その準備として基本格子系 h_0 より 1/2 格子系 h $(=h_0/2)$ への移行について考察する．このとき，両格子系が直接接触する場合には混乱するので，図 3 に示される中間格子帯を緩衝帯として設ける．ここでは，格子点 a では十字形，b ではX字形の格子点配置を採用すればよい．必要があればさらに 1/4, 1/8, ・・・ と

格子間隔をさらに短縮できる．なお格子点が境界線上にのらないときは直交不等間隔差分を行えばよい(図3)．

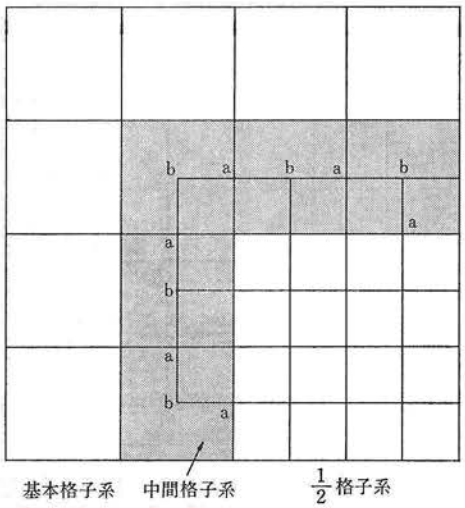


図 3 格子間隔の変更
Fig. 3 Refinement of meshes

3.2 3次精度展開格子系

2次精度展開にて精度が十分でない場合は3次精度展開(格子点10個)を行うことができる．格子点の配置が正則であるためには，これらの点の1つが1つの3次曲線，または1つの直線と2次曲線，または3本の直線の上にのらないことが必要である．次に，2つの格子点配置について考察する．

例 1 A格子点配置 図4 (a)に示す格子配置で，P_8 は3つの直線 $L_1(y=0)$, $L_2(x=0)$, $L_3(x+y=0)$ の外にある．したがって，この格子点配置は正則である．当然，8格子点はこの位置の他に，3直線にのらないどの格子点に移してもさしつかえない． Laplace 方程式は

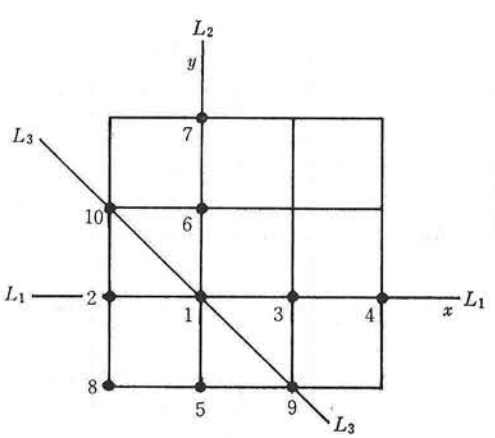
$$u_{xx}+u_{yy}=(-4u_1+u_2+u_3+u_5+u_6)/h^2 \qquad (3\text{-}5)$$

で表され，2次精度の場合と同じである．

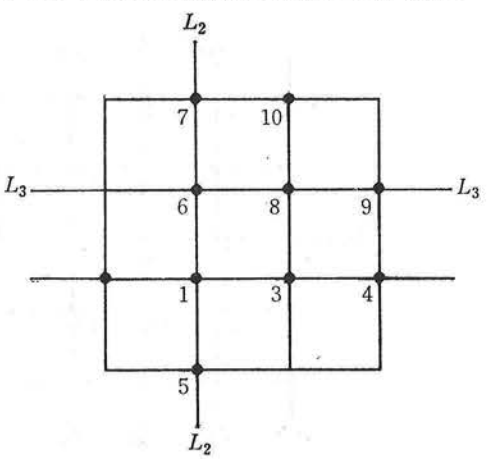
例 2 B格子点配置 この格子点配置 (図4 (b))では $L_1(y=0)$, $L_2(x=0)$, $L_3(y=h)$ にのらない格子点 P_{10} が配置されている．したがって，この格子点配置も正則である．Laplace 方程式は例1の場合と同じで，(3-5)で表される．このように，3次精度展開では大きな精度の改善は期待できない．

3.3 4次精度展開格子系

2階微分方程式の高精度差分化，または板曲げや N/S 方程式等の4階微分方程式の差分化には4次精度展開による格子系が利用される．ここでは，図5に示される代表的な格子点配置について考察する．この格子点配置では，P_1〜P_{14} は $L_1(y=0)$, $L_2(x=0)$, $L_3(y=h)$, $L_4(y=-h)$ 上にのり，P_{15} は除かれている．したがって，これは正則である．ここでは Laplace 方程式と2重 Laplace 方程式，およびそれぞれの差分関係の図(図6 (a), (b))を



(a) A格子点配置



(b) B格子点配置

図 4 2つの格子点配置
Fig. 4 Difference families A and B

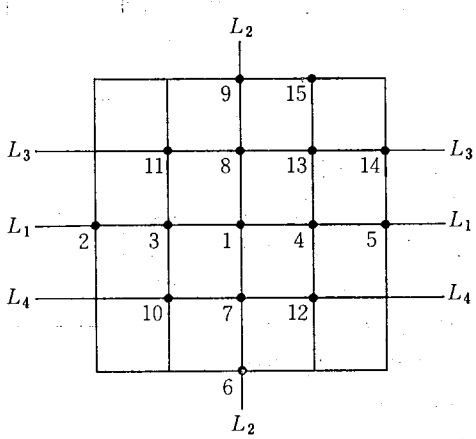
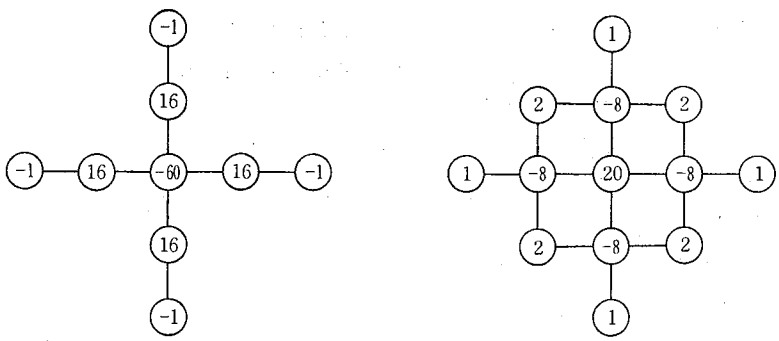


図 5　4次精度展開による代表的な格子点配置
Fig. 5　Typical difference family for 4th order approximation



（a）　Laplace 式の差分関係　　（b）　2重 Laplace 式の差分関係

図 6　差分関係
Fig. 6　Difference relation for (double) Laplaces equation at interior point

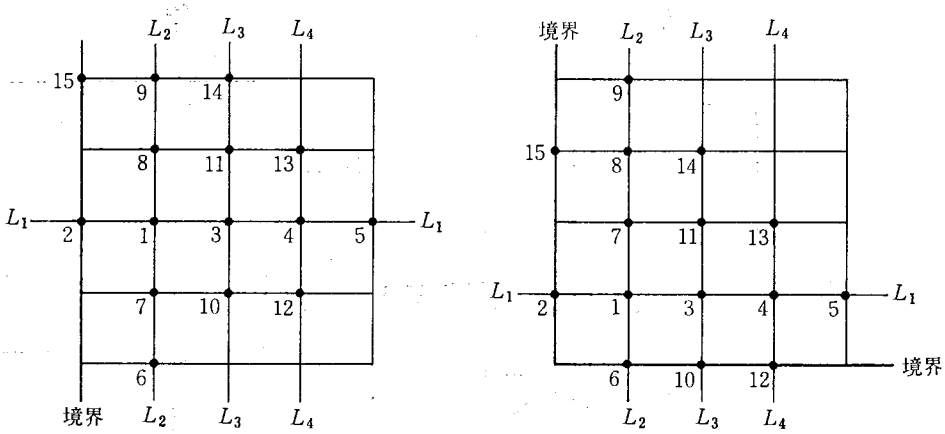


図 7　境界に隣りあう正則な格子点配置
Fig. 7　Difference relation for double potential function

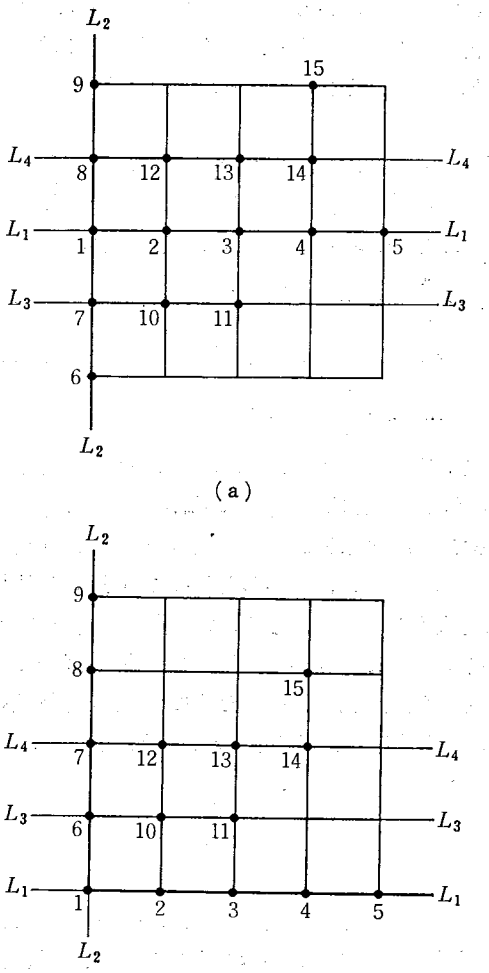


図 8 自由境界の代表的な格子点配置
Fig. 8 Typical difference families on free boundary

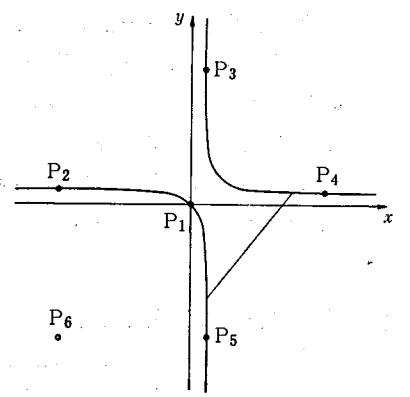


図 9 十字形に近い格子点配置
Fig. 9 An almost orthogonal curvilinear net

示すにとどめる.

$$u_{xx}+u_{yy}=(-60u_1-u_2+16u_3+16u_4-u_5-u_6+16u_7+16u_8-u_9)/12h^2$$

$$u_{xxxx}+2u_{xxyy}+u_{yyyy}=(20u_1+u_2-8u_3-8u_4+u_5+u_6-8u_7-8u_8+u_9+2u_{10}+2u_{11}+2u_{12}+2u_{13})/h^4$$

また，境界に隣りあう格子点で，正則な格子点配置は図 7 (a), (b) である.

自由境界では 2 次，3 次微係数の差分化が必要となるので，このときの代表的な格子点配置を示す．図 8 (a), (b), (c) は差分評価格子点 P_1 が境界線（太い線）上に位置するときの代表例で，いずれも 1 つの格子点 P_{15} は 4 本の直線 L_1, L_2, L_3, L_4 からはずれている．したがって，この格子点配置は正則である.

4. 直交曲線座標格子系による差分方程式の導入

前述のとおり，複雑な形の境界を持つ境界値問題の解析も，家族内の格子点配置が正則でありさえすれば，境界条件式の差分化が容易に行える．直交曲線座標系では微視的に直交および等間隔性が保たれているが，格子間隔が有限量となると巨視的に直交および等間

隔性が共に乱される．したがって粗い格子網を用いるときは，たとえ正確な直交曲線座標格子系を用いる場合も格子点配置に再検討を必要とする．とくに流体力学におけるよどみ点のような特異点では等角性が微視的にも満たされなくなる．このような場合，下記の検討を行えばよい．

一般に n 次精度展開式で項数を N とするとき，n 次曲線（n 本の直線群を含む）は $N-1$ 個の点によって決定される．たとえば2次式ならば5個，3次式ならば9個，4次式ならば14個である．ここでは簡単のため2次式について考察する．変数 $u(x, y)$ を

$$u(x, y)=\alpha_1+\alpha_2+\alpha_3+\alpha_4\frac{x^2}{2}+\alpha_5 xy+\alpha_6\frac{y^2}{2}=\sum_1^6\alpha_n f_n(x, y) \tag{4-1}$$

によって表すとき，2次曲線は $u(x, y)=0$ である．$P(x_m, y_n)$ を通る2次曲線は，

$$\sum_1^6\alpha_n f_n(x_m, y_m)=\sum_1^6 f_{mn}\alpha_n=0 \tag{4-2}$$

によって表される．もし6点がすべて2次曲線上にあれば，α_n に関する同次連立方程式 (3-2) が満たされるので，$\alpha_n \neq 0$ 解が存在するためには，$\det f_{mn}=0$ が満たされなければならない．したがって，(1-7) によって α_n を求めることはできなくなる．節点を直交曲線座標格子系の格子点にとるとき，その配置はほぼ十字形となっているので正則であることは間違いない．格子間隔を次第に増大させるとき，とくに特異点の近傍では P_1〜P_5 にのる2次曲線は直交する2本の直線から次第に遠ざかり，他の2次曲線となる．したがって，この曲線が P_6 の近くを通過する可能性があり，そのために格子点配置が不適当となることがある．実際には P_1〜P_5 の配置が十字形にほぼ近ければ，P_6 を図9に示すような位置に配することにより，正則性を保つことができる．

5. 曲線座標格子系による数値実験

前章までに微分方程式の解法として種々の差分化手法を述べた．その中で，とくに新しいアプローチと見られる曲線座標格子系をとりあげ，プログラム開発と数値実験を試みた．理論解との比較が容易なようにポテンシャル流をモデルとした．

5.1 ポテンシャル流の支配方程式

2次元理想気体の運動の支配方程式は次のとおりである．Euler の運動方程式は，

$$\frac{\partial u}{\partial t}+u\frac{\partial u}{\partial x}+v\frac{\partial u}{\partial y}=-\frac{1}{\rho}\frac{\partial P}{\partial x}+f_x,\quad \frac{\partial v}{\partial t}+u\frac{\partial v}{\partial x}+v\frac{\partial v}{\partial y}=-\frac{1}{\rho}\frac{\partial P}{\partial y}+f_y \tag{5-1}$$

であり，連続の式は次のようになる．

$$\frac{\partial u}{\partial x}+\frac{\partial v}{\partial y}=0 \quad (u, v \text{ は } x, y \text{ 軸方向の速度成分，} P \text{ は圧力}) \tag{5-2}$$

流体が非回転であれば，渦度 ω_z は零，すなわち，

$$\omega_z=\frac{\partial v}{\partial x}-\frac{\partial u}{\partial y}=0 \tag{5-3}$$

である．この式は，

$$u=\frac{\partial \phi}{\partial x},\qquad v=\frac{\partial \phi}{\partial y} \tag{5-4}$$

を満足するポテンシャル関数 ϕ の存在を保証する．

式 (5-4) を連続の式 (5-2) に代入すれば，

$$\frac{\partial^2 \phi}{\partial x^2}+\frac{\partial^2 \phi}{\partial y^2}=0 \tag{5-5}$$

が得られ，与えられた境界条件のもとで(5-5)を満足する ϕ を求めることになる．また，(5-2)により，流れ関数 ϕ が存在し，

$$u=-\frac{\partial\phi}{\partial y},\qquad v=\frac{\partial\phi}{\partial x} \tag{5-6}$$

なる関係がある．これを非回転条件式(5-3)に代入すれば，

$$\frac{\partial v}{\partial x}-\frac{\partial u}{\partial y}=\frac{\partial^2\phi}{\partial x^2}+\frac{\partial^2\phi}{\partial y^2}=0 \tag{5-7}$$

となり，(5-5)と同様に Laplace 方程式が得られる．

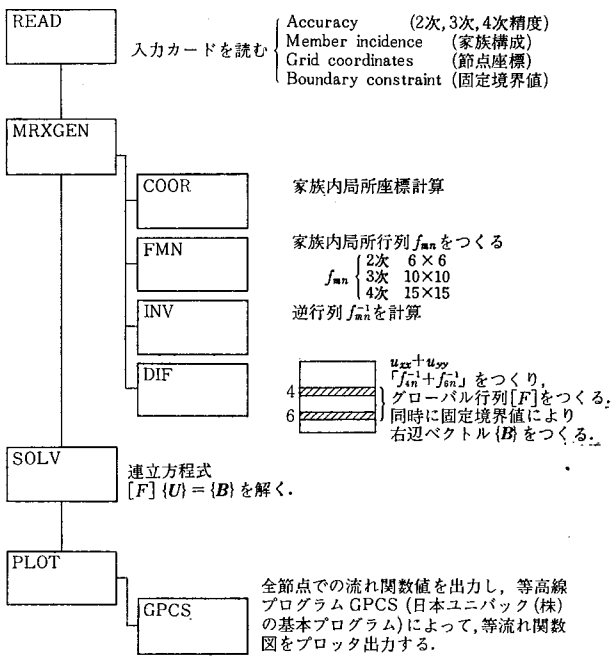


図 10 プログラムの流れ図
Fig. 10 Program flow

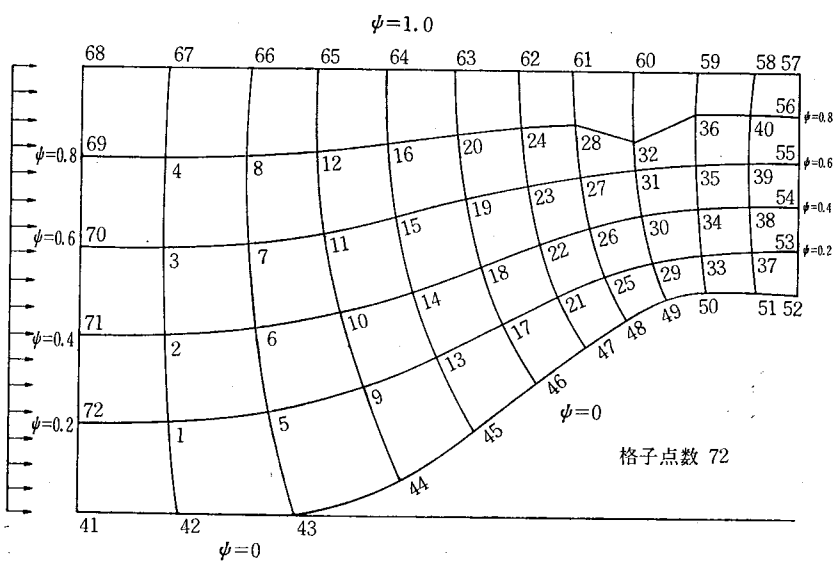


図 11 格子点と境界条件
Fig. 11 Curvilinear net for sample analysis

以上から，ポテンシャル流においては速度ポテンシャル ϕ または流れ関数 ψ に関する Laplace 方程式が支配方程式となる．

5.2 曲線座標格子差分によるポテンシャル流解析プログラム

本プログラムは2次精度(6点差分)，3次精度(10点差分)，4次精度(15点差分)のいずれかを自由に選択可能なようにつくられている．差分評価点を原点とする変数 $u(x, y)$ は第2章で述べたように，次式で表される．

$$u(x, y)=\lfloor f_n \rfloor \{\alpha_n\} \quad (2\text{次精度 } n=6,\ 3\text{次精度 } n=10,\ 4\text{次精度} n=15) \tag{5-8}$$

評価点近傍の節点値は $u_m=u(x_m, y_m)$ と表して

$$\begin{Bmatrix} u_1 \\ \vdots \\ u_m \end{Bmatrix} = \begin{bmatrix} 1 & x_1 & y_1 & \cdots \\ \vdots & \vdots & f_{mn} & \\ 1 & x_m & y_m & \cdots \end{bmatrix} \begin{Bmatrix} \alpha_1 \\ \vdots \\ \alpha_n \end{Bmatrix} \tag{5-9}$$

$$\{u_m\}=[f_{mn}]\{\alpha_n\} \tag{5-10}$$

である．したがって，(5-9)は節点値 u_m を用いて，

$$u(x, y)=\lfloor f_n \rfloor [f_{mn}^{-1}] \{u_m\} \tag{5-11}$$

と表される．すなわち評価点において

$$\left.\begin{aligned} u(x, y)&=\lfloor f_n \rfloor [f_{mn}{}^{-1}]\{u_m\} &&=\lfloor 10\cdots00 \rfloor [f_{1n}{}^{-1}]\{u_m\} &&=f_{1n}{}^{-1}u_n \\ u_x(x, y)&=\lfloor f_{n,x} \rfloor [f_{mn}{}^{-1}]\{u_{mn}\} &&=\lfloor 01\cdots00 \rfloor [f_{2n}{}^{-1}]\{u_m\} &&=f_{2n}{}^{-1}u_n \\ u_y(x, y)&=\lfloor f_{n,y} \rfloor [f_{mn}{}^{-1}]\{u_{mn}\} &&= \cdots &&=f_{3n}{}^{-1}u_n \\ u_{xx}(x, y)&=\lfloor f_{n,xx} \rfloor [f_{mn}{}^{-1}]\{u_{mn}\} &&= \cdots &&=f_{4n}{}^{-1}u_n \\ u_{yy}(x, y)&=\lfloor f_{n,yy} \rfloor [f_{nm}{}^{-1}]\{u_{mn}\} &&= \cdots &&=f_{6n}{}^{-1}u_n \end{aligned}\right\} \tag{5-12}$$

となる．ここに，$u_x=\partial u/\partial x$, $u_{xx}=\partial^2 u/\partial x^2$, $\cdots$ であり，$\lfloor f_{4n}^{-1} \rfloor$ は行列 $[f_{mn}^{-1}]$ の4行目を表すものとする．したがって，ポテンシャル流を表す Laplace 方程式は次のようになる．

$$u_{xx}+u_{yy}=\lfloor f_{4n}^{-1}+f_{6n}^{-1} \rfloor \{u_m\}=0 \tag{5-13}$$

解析領域のすべての内部点について上式の方程式をつくり，連立方程式をつくる．さらに境界条件を考慮して，方程式を解けば節点での流れ関数値が得られる．

プログラムの流れ図を図10に示す．

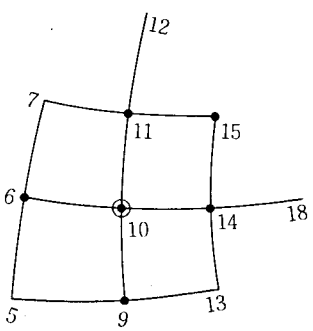


図 12 節点10の家族
Fig. 12 10 point difference family

5.3 曲線座標格子差分によるポテンシャル流の解析

例 1 境界の一部が曲線となる一様流 まず，プログラムのテスト的意味合いを含め，簡単な一様流について解析した．図11には，境界条件によって解析領

* FAMILY TABLE **

1	2	5	41	42	72
2	1	3	6	71	72
3	2	4	7	69	70
4	3	8	67	68	69
5	1	6	9	42	43
6	2	5	7	9	10
7	3	6	8	10	11
8	4	7	12	65	66
9	5	10	13	44	45
→ 10	6	9	11	14	15
11	7	10	12	15	16
12	8	11	16	64	65
13	9	14	17	45	46
14	10	13	15	18	19
15	11	14	16	19	20
16	12	15	20	63	64

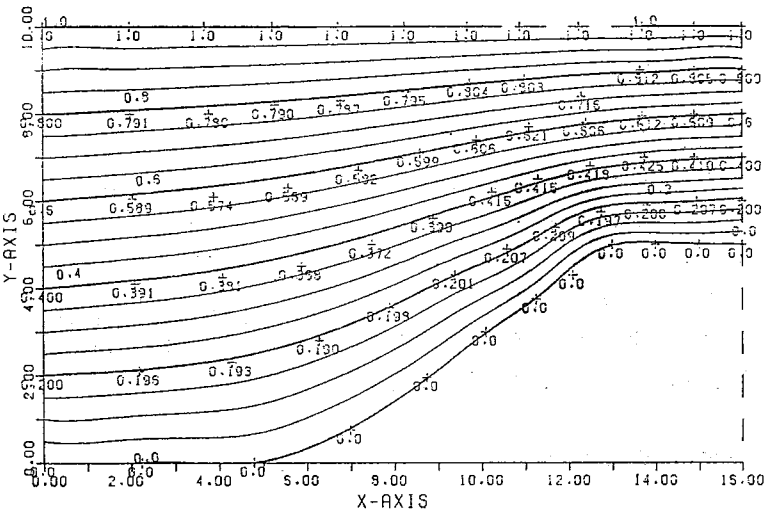
(a) 2次精度

* FAMILY TABLE **

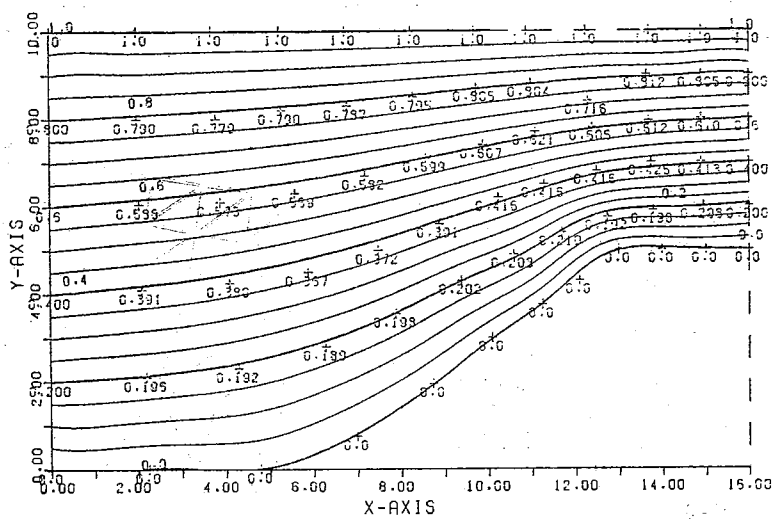
1	41	72	71	42	2	3	43	5	9
2	72	71	70	1	3	4	5	6	10
3	71	70	69	2	4	67	6	7	11
4	68	67	66	69	8	12	70	3	2
5	42	1	2	43	6	7	44	9	13
6	1	2	3	5	7	8	9	10	14
7	2	3	4	6	8	46	10	11	15
8	67	66	65	4	12	16	3	7	6
9	43	5	6	44	10	11	45	13	17
→ 10	5	6	7	9	11	12	13	14	18
11	6	7	8	10	12	65	14	15	19
12	66	65	64	8	16	20	7	11	10
13	44	9	10	45	14	15	46	17	21
14	9	10	11	13	15	16	17	18	22
15	10	11	12	14	16	64	18	19	23
16	65	64	63	12	20	24	11	15	14

(b) 3次精度

図 13 家族構成の一部
Fig. 13 2nd and 3rd order approximation



(a) 2次精度 6点差分



(b) 3次精度 10点差分

図 14 解析結果

Fig. 14 Solutions by 2nd order approximation (6 point difference) and 3rd order approximation (10 point difference)

域の周辺で与えられた流れ関数値，および曲線座標格子点を示す．また，2次精度と3次精度の家族構成の例を図12および図13(a), (b)に示す．これらの図からわかるように，中心節点10を中心に0節点は6点差分にとられる節点，これらを含めたすべての節点は10点差分にとられている．2次精度，3次精度の解析結果を図14(a), (b)に示す．

例 2 一様流に吹出しがある流れの場(有限要素法との比較) 図15に示すように，x 方向に速度 u を持つ一様流があり，その中に強さ q の吹出しがあるものとする．この流れの場について，速度ポテンシャル $\varphi(x, y)$，流れ関数 $\psi(x, y)$ は

$$\varphi = ux + q \log r, \qquad \psi = uy + q\theta \tag{5-14}$$

で表される．ここで，$r=\sqrt{x^2+y^2}$, $\theta=\tan^{-1}(y/x)$ である．また，$\xi=x/l$, $\eta=y/l$, $\rho=r/l$

とおくと，(5-14)のそれぞれは

$$\left.\begin{aligned}\frac{\varphi}{nl}&=\xi+\frac{g}{nl}(\log l+\log\rho)\\ \frac{\psi}{nl}&=\eta+\frac{g}{nl}\theta\end{aligned}\right\}\quad(5\text{-}15)$$

となる．$g/nl=1$ とし，改めて ξ,η,ρ をそれぞれ x,y,r として

$$\varphi=x+1+\log r,\quad \psi=y+\theta\qquad(5\text{-}16)$$

としても，一般性は失われない．

式(5-16)より $r=e^{\varphi-x-1}$ すなわち $r^2=x^2+y^2=e^{2(\varphi-x-1)}$ である．したがって

$$y^2=e^{2(\varphi-x-1)}-x^2.\qquad(5\text{-}17)$$

同様に(5-16)より，$\tan\theta=y/x=\tan(\psi-y)$ となるので，

$$\therefore\ x=y/\tan(\psi-y)\qquad(5\text{-}18)$$

が得られる．

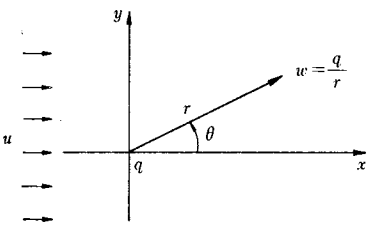


図 15　一様流に吹出しのある流れの場
Fig. 15　A uniform stream in a field of flow with a source

式(5-16)から計算される境界条件を使い，図16の曲線格子によって解析を試みた．図17は比較のために試みた有限要素法による解析のための要素分割図を示す．また，理論解

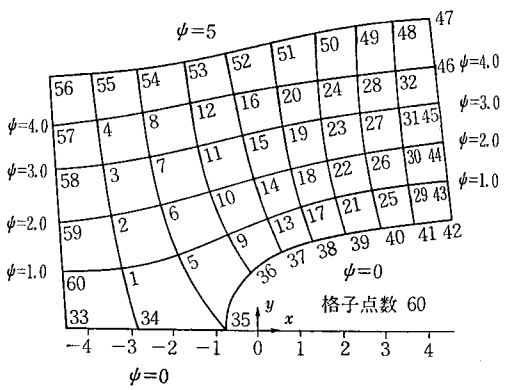


図 16　格子点と境界条件
Fig. 16　Net and boundary for sample

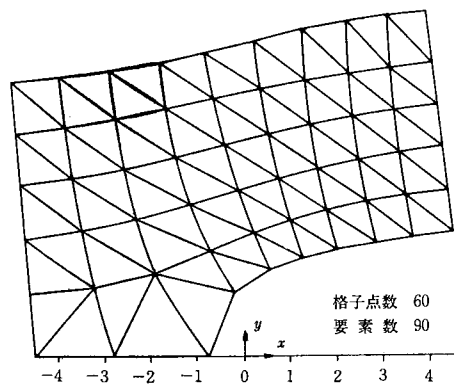
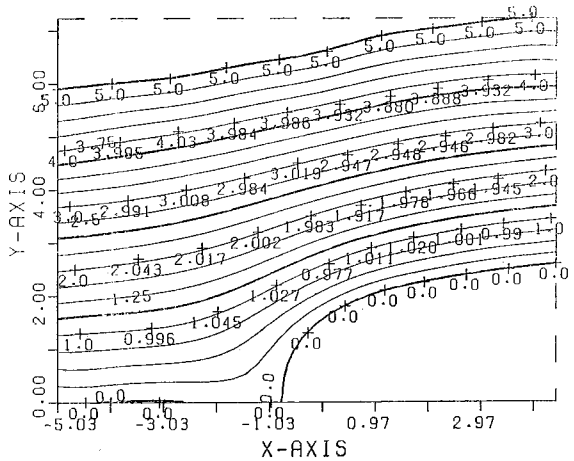


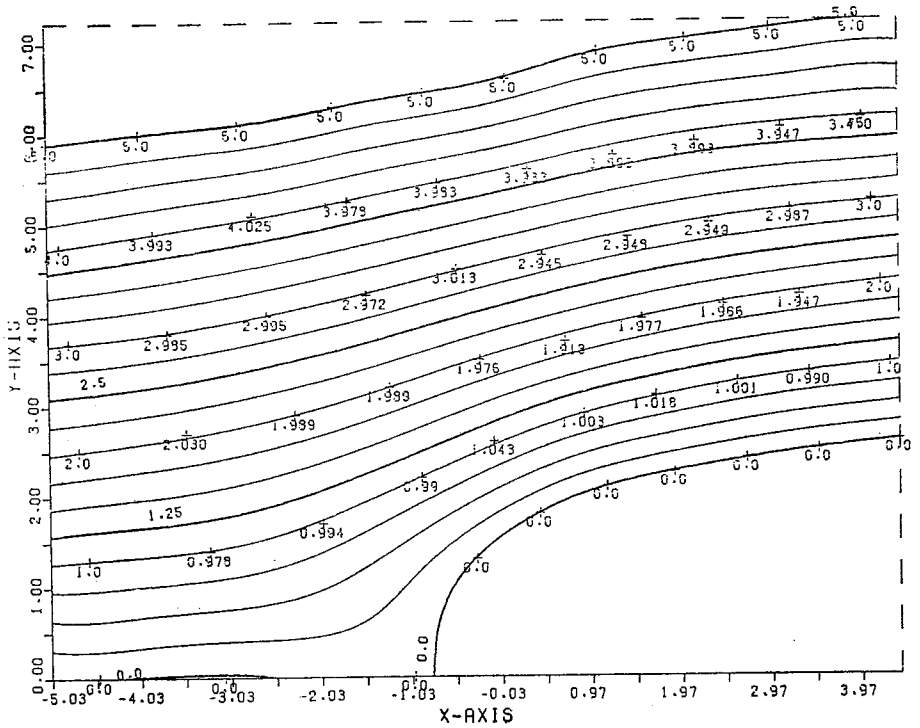
図 17　有限要素法解析のための要素分割図
Fig. 17　Finite element distribution for FEM analysis

表 1　理論解 (Exact)，差分解 (D)，有限要素法 (FEM) の数値比較
Table 1　Numerical comparison between Exact solution and results by Difference method and FEM

ノード	Exact	D	FEM	ε[%] D	ε[%] FEM	ノード	Exact	D	FEM	ε[%] D	ε[%] FEM
1	0.993	0.996	0.978	0.30	1.51	10	2.002	2.017	1.989	0.0	0.65
2	2.043	2.043	2.030	0.0	0.64	11	2.991	3.008	2.972	0.23	0.64
3	3.001	2.991	2.985	0.33	0.53	12	3.992	3.984	3.978	0.20	0.35
4	3.995	3.995	3.993	0.0	0.05	13	0.991	0.977	1.043	1.41	5.24
5	0.996	1.043	0.994	4.7	0.2	14	2.001	1.983	1.976	0.90	1.25
6	2.000	2.011	1.989	0.85	0.55	15	3.040	3.019	3.013	0.69	0.89
7	3.006	3.008	2.995	0.07	0.37	16	4.007	3.986	3.983	0.52	0.60
8	4.024	4.030	4.025	0.15	0.03	17	1.028	1.011	1.003	1.65	2.4
9	1.018	1.027	0.990	0.88	2.75	18	1.942	1.917	1.913	1.29	1.49



（a） 2次精度　6点差分



（b） 有限要素法

図 18　一様流に吹出しがある流れの場の解析結果

Fig. 18　Solutions by 2nd order approximation (6 point difference) and FEM for the field of a uniform flow with source

(Exact), 差分解 (D), 有限要素法解 (FEM) の数値比較 (ε) を表 1 に示す．理論解は (5-18), $\psi=y+\theta$, $\theta=\tan^{-1}(y/x)$ による．

2次精度差分と有限要素法の解析結果を図18(a), (b) に示す．

例 3　吹出し，吸込みのある流れの場（2段分割の試み）　一般に解の精度を高めるために，差分法では多段メッシュを採用する．しかし，これはプログラムが複雑になり，簡

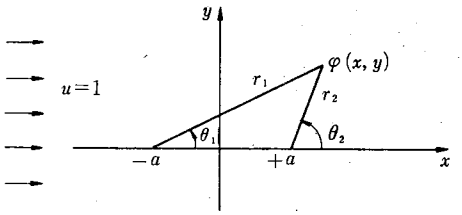


図 19　一様流に吹出し，吸込みのある流れの場
Fig. 19　A uniform stream in a field of flow with a source and a sink

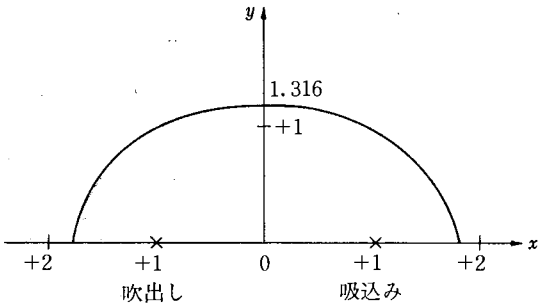
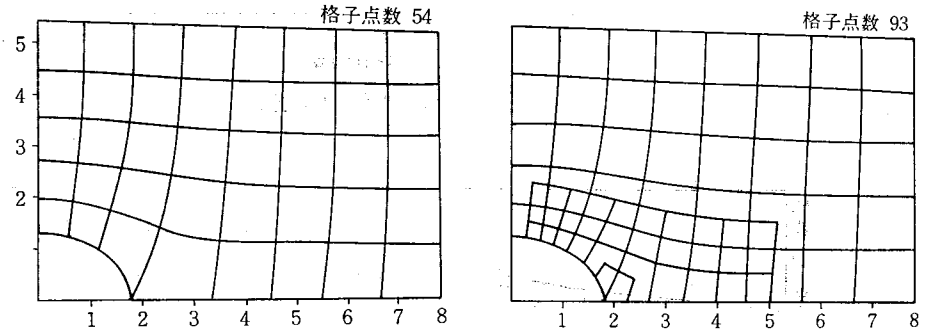


図 20　$\phi=0$ の流線
Fig. 20　Stream line for $\phi=0$



（a）　格子点数 54　　（b）　格子点数 93

図 21　格子点
Fig. 21　Used net

単には採用できない．ここでは有限要素法なみの手軽さで格子点の粗さが自由に変えられることを示す．

図19 に示すように，一様な流れの中に等しい強さの吹出し，吸込みが流れの方向にあるときの速度ポテンシャル φ と流れ関数 ϕ を求める．標準化により，φ と ϕ はそれぞれ

$$\varphi=ux+q\log r_1-q\log r_2=ux+q\log(r_1/r_2),\quad \phi=y+\theta_1-\theta_2 \tag{5-19}$$

となる．$r_1, r_2, \theta_1, \theta_2$ は各々 $r_1=\sqrt{(x+a)^2+y^2}$, $r_2=\sqrt{(x-a)^2+y^2}$, $\theta_1=\tan^{-1}[y/(x+a)]$, $\theta_2=\tan^{-1}[y/(x-a)]$ である．(5-19)から，次式がそれぞれ得られる．

$$y^2=2ax\cot h(\varphi-\varphi_0-x_0)-(x^2+a^2)\qquad(\varphi_0\text{ は積分定数})\tag{5-20}$$

$$x^2=a^2+\frac{2ay}{\tan(y-\phi)}-y^2 \tag{5-21}$$

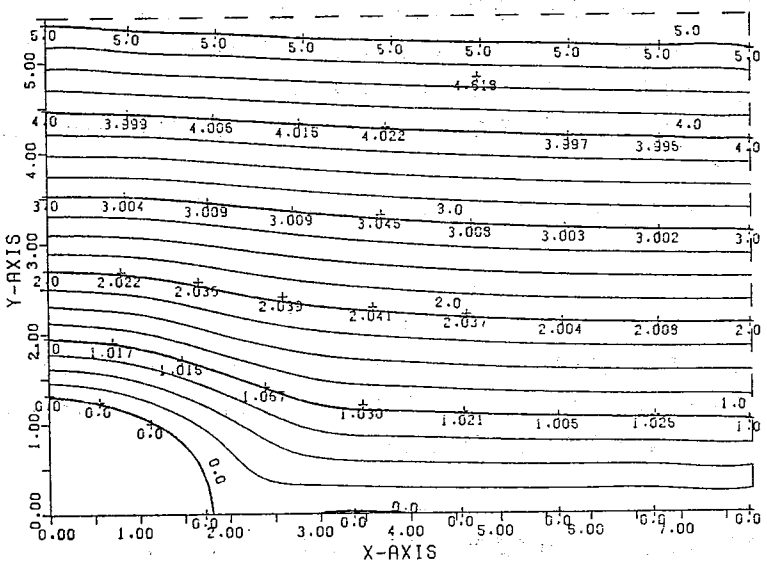
ここで，流れ関数 $\phi=0$ に相当する式を求める．(5-21)で $\phi=0$ として，$y=0$ とするとき，xは不定形となる．ここで $y\neq0$ の解を求めると

$$x^2=a^2+2a\frac{y}{\tan y}-y^2 \tag{5-22}$$

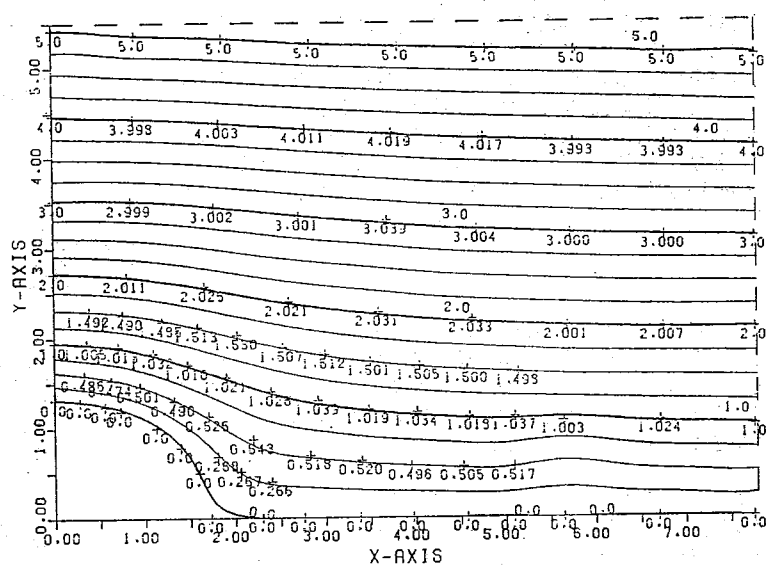
となる．上式の $y\to0$ の極限値 $x^2=a^2+2a$，すなわち $y\neq0$ の解がx軸を切る点は $x_0=\pm\sqrt{a(a+2)}$ である．次に，$\phi=0$ の曲線が y 軸を切る点は，(5-20)より $a=y_0(-1/\tan y_0+1/\sin y_0)$ と求められる．

式(5-20)および上で求めた x_0, y_0 により，$a=1$ としたときの $\phi=0$ の流線が計算される．図 20 は $\phi=0$ の流線を示す．

式(5-20)により計算された $\phi=0$ の流線を境界として，図21(a)，(b)の格子点により解



（a）　2次精度



（b）　2段メッシュ

図 22　吹込み・吹出しのある流れの場の解析結果

Fig. 22　Solutions both by 2nd order approximation, and with double-stage net for the field of flow with source and sink

析を試みた．これら2次精度差分による解析結果を図22(a), (b)に示す．

5.4　数値実験に関するまとめ

提案された曲線座標格子系差分解法が十分な精度を保ち，実用になるか否かを調べるため，ポテンシャル流をモデルとして選んで理論解あるいは他手法と比較した．その結果，十分実用になることが認められた．今後 Navie Stokes 方程式，板曲げモデルなどについても報告する予定である．

プログラムを書いてみて，気がついたこの手法の優位性は

1) 現象を表現する微分方程式から出発でき，理解しやすい．
2) プログラムにいくつもの精度次数が組み込めるので，モデルに応じて精度が選べる．
3) 多段メッシュに対する処理をプログラムで考慮しないですむ．
4) プログラムの骨組みをつくっておけば，部分的修正で種々の微分方程式に対応できる．
5) 家族構成節点の選択に当たり制限がゆるい(節点番号の順番が自由)．

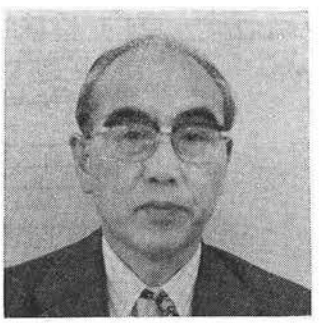
となる．

6. おわりに

以上，新しい差分法とその数値実験について報告した．提案の差分法が有限要素法のように普遍的に利用されていないことは確かであるが，今後さらに科学，工学の分野で十分活用されるよう研究を進める予定である．

参考文献 [1] 池田昌弘，他，有限要素法の応用，日本鋼構造協会，1975.
[2] W. H. エイムズ著，三村昌泰，小西芳雄訳，工学における非線形偏微分方程式 I 下，産業図書，1978.

執筆者紹介 藤野　勉 (Tsutomu Fujino)
明治45年生，昭和11年東京帝国大学理学部物理科卒業，同年三菱重工(株)入社．主に応力・振動・流体力学などの解析法(有限要素法を含む)の研究に従事．32年，工学博士号を取得．47年，同社技術本部顧問となる．また，48年より東海大学工学部教授，52年依嘱教授を歴任．51年より日本ユニバック(株)の技術顧問となり，現在に至る．「コンピュータによる構造工学講座 II-4-B—熱伝導と熱応力」(培風館，1972) などの著書がある．



渡部義維 (Yoshizumi Watanabe)
昭和14年生，44年早稲田大学大学院理工学研究科(博士)修了．同年日本ユニバック(株)入社，現在に至る．応用ソフトウェア部技術計算グループにおいて，有限要素法プログラム・サービスなどに従事．物理学会会員．

編集者注 本論文と同趣旨のものが1980年12月12日のユニバック研究会第11回技術計算分科会・構造解析事例発表会において「直交曲線格子座標系差分法の提案と非圧縮粘性流体の数値実験」として発表されている．

回路解析プログラム CIRCUIT の特徴と数値計算技法

Technical Report of CIRCUIT a New Electronic Circuit Analysis Program

長島 毅

要 約 CIRCUIT は，本稿に述べるような最新の算法の採用により，電気電子回路解析のための最も進んだ設計ツールとして1977年にレベル 1R1 が完成した．その後，機能と性能の改良を加えて，レベル 2R1D に至っている．1980年12月現在でのユーザ数は，テスト使用中を含めて国内外で20数社を数えている．また，対象回路も解析種類も非常に多岐にわたっており，CIRCUIT はその汎用性を発揮して，有用な設計ツールとして定着しつつある．また，本プログラムで用いた数値計算法の具現化の経験は，今後行われるであろう他の技術計算用ソフトウェアの開発にも非常に有益な情報として，役立つことと思われる．なお，SFTN(日本ユニバック(株)で開発した ASCII FORTRAN のプリコンパイラ) による構造化プログラミングによって，CIRCUIT の開発と保守の費用は低減されたことを付記しておく．

Abstract This paper describes the notable features of "CIRCUIT", which is developed at Nippon Univac Kaisha Ltd. as a new electric and electronic circuit analysis program. Furthermore, several numerical computation algorithms of CIRCUIT are introduced and evaluated for the sake of characterization of this program.

ASTAP released by IBM has been appreciated for its flexibility, statistical analysis feature and numerical stability. Such advanced circuit analysis programs have become indispensable tools for IC's or LSI's design with increasing integration density and variety of nonlinear characteristics of semiconductor devices.

Under these circumstances, we began to develop a new software in 1975, and CIRCUIT was accomplished in 1977 as a new analysis program. All the functions of ASTAP are included in CIRCUIT, and some attractive functions are supplemented to assure advanced network analysis and design.

1. はじめに

エレクトロニクスの急激な進歩は，IC，LSI のような高い集積度を持つ電子回路を生み出した．IC，LSI の設計者あるいはこれらを応用した電子機器の設計者にとって，コンピュータによる回路シミュレーションが，設計の精度と効率の向上に必要不可欠な手段として定着しつつある．またその適用範囲も大規模化し，かつ続々と現れる素子特有の非線形電流電圧特性も含めていかなければならなくなっている．CIRCUIT は，このような背景のもとで，日本ユニバック(株)の提供する汎用の電気電子回路解析ソフトウェアとして開発されたプログラムである．その仕様の設定に当たっては，多面的な利用形態に柔軟に対処できるように考慮して，以下の特徴を持たせることとした．

- 7種類の基本素子に任意の特性式を持たせて任意の等価回路が記述可能となる．
- 大規模回路の記述が容易になるように入れ子ができるサブ回路マクロ機能を導入する．
- 同一の回路定義データに直流(DC)，過渡(TR)，交流(AC)の3種類の解析モードを指定できるようにする．
- ケース・スタディ機能を持たせる．
- 統計的手法による信頼性解析機能を持たせる．

・回路パラメタの最適化機能を持たせる.
・出力機能を豊富にして，グラフィック・ディスプレイやXYプロッタ等に出力できるようにする.
・システム等価回路ライブラリを設ける.

このような要求仕様によってCIRCUITをソフトウェアとして具現化する過程で，まず直面した問題は，任意の非線形従属特性を含む任意規模の回路モデルを対象として直流，過渡，交流の領域で高精度でかつ高能率に数値解析を行う数値計算技法をどう選択するかであった．種々の数値計算技法を検討した結果，次のような計算技法を採用した.

・陰積分公式による過渡解析
・擬似過渡解析法による直流解析
・小信号解析による非線形特性を含む回路の交流解析
・上記の各解析で共通に利用される疎性を利用した擬似コードによる行列計算

本稿ではCIRCUITプログラムの特徴とこれら算法の持つ特徴およびソフトウェアとしての算法と改良点について述べる.

2. CIRCUIT の特徴

CIRCUITプログラムの構成は図1に示すように3つの実行モジュールと2つのライブラリからなっている．目的モジュールは与えられた回路モデルの解析効率をもっとも高めるように翻訳モジュールから生成される．出力モジュールは目的モジュールで計算された解析結果の出力エディタである．プログラム・ライブラリは目的モジュールの連結編集のために使用する．モデル・ライブラリは必要に応じて等価回路のカタログを用意するために使用する.

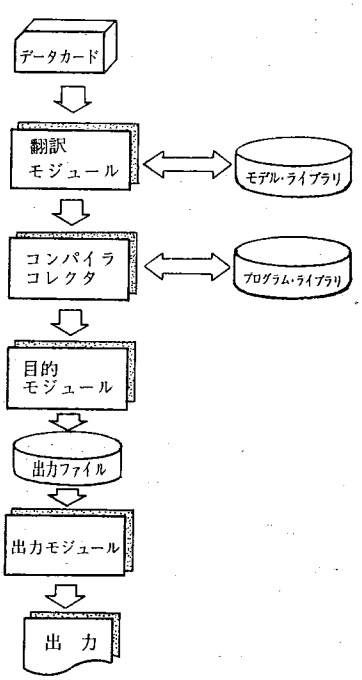


図 1 CIRCUIT のシステム概要
Fig. 1 Process chart of "CIRCUIT" system

CIRCUIT は次のような6つの機能を持っている．

第1は，5つの基本解析モード(DC, TR, AC, DC/TR, DC/AC) を同一のモデル記述に対して使用することができる点である．直流 (DC) 解析は指定したバイアス条件のもとで回路の安定状態の解を導く．過渡 (TR) 解析は時間的に変動する入力に対する回路の過渡応答を解析するモードである．交流 (AC) 解析は与えられた周波数領域の周波数応答を解析するモードである．直流・過渡 (DC/TR) モードでは直流解を過渡解析の初期条件として自動的に連結する．直流・交流 (DC/AC) モードでは，交流解析において直流解を非線形素子特性の線形化のための動作点として利用する．

第2は，3,000 以上の基本素子を1つの回路モデルに含めることができるということである．このような大規模なモデルは IC や LSI の設計において必要である．大規模な問題において高速度の計算を行うためには，高度な疎行列算法が必要であるが，われわれが採用して改良した疎行列算法については後に詳述する．

第3は，任意の非線形素子特性が FORTRAN 的な算術代入文あるいはデータ点のテーブル関数によって表現が可能である点である．このような柔軟なモデル記述は図1に示した FORTRAN のコンパイレーション過程を含む目的モジュールの生成というシステム構成によって実現可能となった．このようにして定義された任意の非線形を含む回路網方程式の直流解を，発散させないよう安定して求められるために，“擬似過渡解析”と呼ばれる反復法を通常の Newton-Raphson 法と組み合わせて使用している．

第4は，サブモデル・マクロ定義機能が，IC メモリ等のように反復するサブパターンを持つ回路の表現に非常に役立つ点である．また，モデル・ライブラリの中にあらかじめ汎用性の高いサブモデル定義を登録しておくと，あとで，再定義することなくそれを自由に参照できるようになっている(図2参照)．

第5は，既知の回路素子特性の確率分布からモンテカルロ法による統計解析を行って，システムの出力パラメタの確率分布が求められる点である．この機能は，従来よく行われ

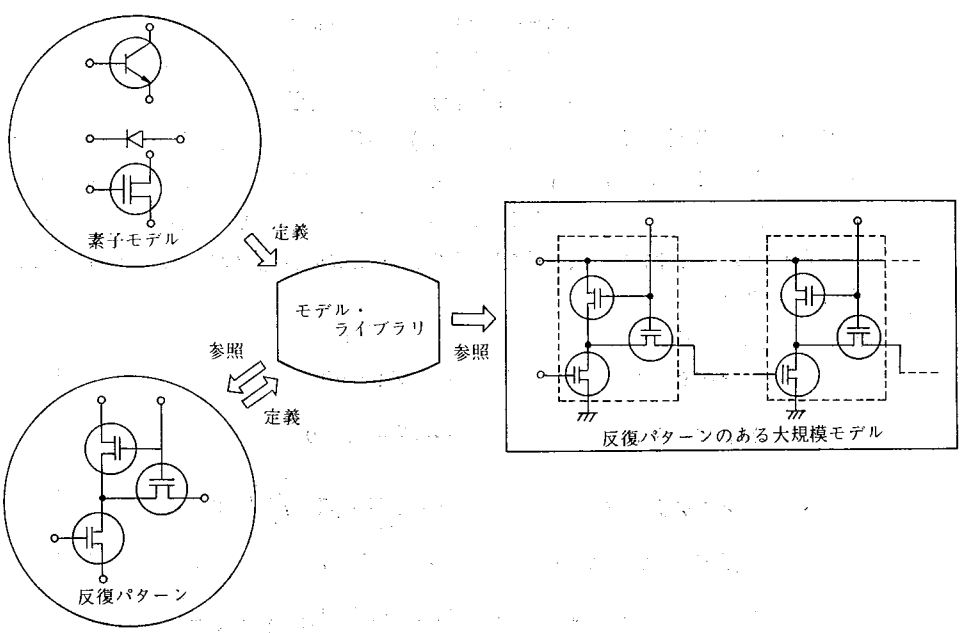


図 2　サブモデルのマクロ定義
Fig. 2　Submodel macro definition

てきたワーストケース解析に比較して，非線形回路の信頼性を検査するのに適している．統計解析で得られる結果は指定した出力パラメタのヒストグラム，過渡ないし周波数応答曲線の包絡線プロット，あるいは分布パラメタ間のスキャッタ・グラムとして観察できる．

第6は，5つの基本解析モードの中の任意のモードに対して最適化解析を指定することができることである．系の応答の“望ましさ”を定量的に表す評価パラメタをモデル記述とともに定義する．つづいて Fletcher-Powell 法によって，この評価パラメタを極小にする調節パラメタの値を求めることができる．この機能は出力レベルをある値にセットしたり，電力損失を最小にする回路網を設計したりするのに有効である．

3. CIRCUIT で用いる数値計算技法

本章では CIRUCIT に採用した計算技法についての基本的なものを考察する．

3.1 回路方程式の立式

CIRCUIT 言語によって表現された回路モデルを用いて，CIRCUIT が立式する回路方程式について概説する．いま与えられた回路は6種類の基本2端子素子(回路枝)と相互誘動係数の組合せで表現される．したがって回路解析の目的は N 元の未知ベクトル $\boldsymbol{E}, \boldsymbol{V}, \boldsymbol{I}$ の値を数値的に求めることに帰着する．

ここで N は回路枝の数，$\boldsymbol{E}$ は各回路枝の特性値(抵抗値等)，$\boldsymbol{V}$ は各回路枝の端子間電位差，$\boldsymbol{I}$ は各回路枝の電流である．N 個の回路枝は節点との接続関係から，N_T 個の tree-branch 群と，N_L 個の link-chord 群に分割できる．N_T は節点数 -1 で，$N_L=N-N_T$ である．$\boldsymbol{V}_T$, $\boldsymbol{I}_T$ を tree-branch 群の電圧，電流とし，$\boldsymbol{V}_L$, $\boldsymbol{I}_L$ を link-chord 群の電圧，電流とする．このとき，$\boldsymbol{E}$, $\boldsymbol{V}=[\boldsymbol{V}_T, \boldsymbol{V}_L]$, $\boldsymbol{I}=[\boldsymbol{I}_T, \boldsymbol{I}_L]$ の間の関係は次の連立方程式で記述される．

$$\boldsymbol{I}_T+Q\boldsymbol{I}_L=\boldsymbol{0} \qquad (N_T \text{ 元}) \qquad (3\text{-}1\,a)$$

$$\boldsymbol{V}_L-Q^T\boldsymbol{V}_T=\boldsymbol{0} \qquad (N_L \text{ 元}) \qquad (3\text{-}1\,b)$$

$$\boldsymbol{f}(\boldsymbol{I}, \boldsymbol{V}, \boldsymbol{E})=\boldsymbol{0} \qquad (N \text{ 元}) \qquad (3\text{-}1\,c)$$

$$\boldsymbol{E}=\boldsymbol{g}(\boldsymbol{I}, \boldsymbol{V}, t) \qquad (N \text{ 元}) \qquad (3\text{-}1\,d)$$

式(3-1 a)は Kirchhoff の電流則，(3-1 b)は Kirchhoff の電圧則である．Q はカット・セット行列と呼ばれる $N_T\times N_L$ の0と ±1 とを要素にする係数行列で，回路枝のトポロジカルな結合状態を表現する．(3-1 c)は Ohm 則で各回路枝の種類によって次のようになる．(ただし，X_j は J の値を決定するのに必要な回路枝の電圧値(V_j)または電流値(I_j)を表す．)

$$\left.\begin{array}{ll}
\text{抵　　抗} & V-RI=0 \\
\text{コンダクタンス} & -GV+I=0 \\
\text{キャパシタンス} & -C\dfrac{dV}{dt}+I=0 \\
\text{インダクタンス} & V-L\dfrac{dI}{dt}-\sum M_i\dfrac{dI_i}{dt}=0 \\
\text{電圧源(独立な場合)} & V-E=0 \\
\text{電流源(従属性のある場合)} & \left(I-\sum_j\dfrac{\partial J}{\partial X_j}X_j\right)-\left(J-\sum_i\dfrac{\partial J}{\partial X_i}X_i\right)=0
\end{array}\right\} \qquad (3\text{-}2)$$

式(3-1 d)は回路枝パラメタの値の決定式である．CIRCUIT では FORTRAN の算術式

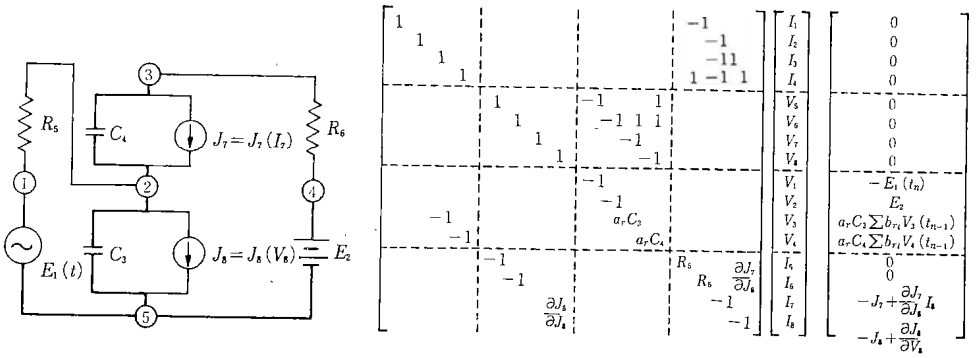


図 3　ハイブリッド形の回路方程式
Fig. 3　Hybrid network equation

や外部関数を用いた任意の従属関係が定義できるから，これも連立して解かなければならない．単純な回路の例について立式した例を図 3 に示す．

3.2 Stiff 積分法による過渡解析

式 (3-1) から過渡応答を導くために，時間微分項 (dx/dt) を Gear の r 次の数値積分公式で置き換える．

$$\frac{dx}{dt}(t_n)=a_r\left[x(t_n)-\sum_{k=1}^{r}b_{rk}x(t_{n-k})\right] \tag{3-3}$$

ここで x は電流値 I，または電圧値 V である．

電子回路は時定数のオーダが大きくばらついて混在する，いわゆる Stiff 系である．Stiff 系の応答は，図 4 に示すように過渡領域と Stiff 領域とに分離できる．

Stiff 領域では，出力は表面上非常になめらかで安定しているが，通常の数値積分計算技法では発散性の不安定さがあるため適当な解が得られない．この現象は小さな時定数の存在によって引き起こされる微係数の数値的な振動が原因となっている．

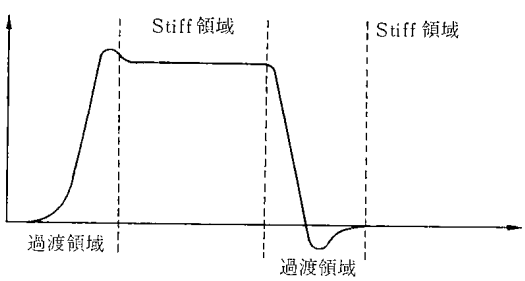


図 4　Stiff 系
Fig. 4　Stiff system

Gear の公式はこの困難を克服できる陰型多段可変次元可変時間きざみ幅方式の数値積分公式である．

式 (3-3) は陰 (implicit) なタイプだから，$x(t_n)$ の値を求める数値積分では同時に未知数である $dx(t_n)/dt$ の値を必要とする．したがって陰な数値積分は，$x(t_n)$ が $dx(t_{n-1})/dt$ から代入演算だけで求められるような陽 (explicit) なタイプの積分よりも 1 ステップに要する計算量が大きいが，Stiff 領域において十分大きなステップ・サイズ $\Delta t=t_n-t_{n-1}$ をとることができる．これに対して，陽な積分のステップ・サイズは Stiff 領域で数値の発散をさけるために非常に小さくしなければならない．そのために，陰なタイプに比べ総計

算量がはるかに大きくなってしまう。

式(3-4)の次数 r は，なめらかな連続部分では高次方程式を使い，急峻に変動する部分や不連続な部分では低次方程式を使うように自動的に調節される．ステップ・サイズ Δt も積分の打切り誤差を許容範囲内におさめるように自動的に調節される．打切り誤差は，予測子

$$x_p(t_n)=\sum_{i=1}^{r} g_{ri}x(t_{n-i}) \tag{3-4}$$

と修正子，つまり現在のステップ・サイズにおける(3-3)の解との間の差から推定することができる．もし推定打切り誤差が許容範囲を越えるならば，その積分ステップは無効となり，再調整した小さいステップ・サイズで積分がやりなおされる．(3-3)の係数 a_r, b_{rk} と(3-4)の係数 g_{rk} は固定されておらず，各時点における過去のステップ・サイズの履歴から算出される．

Gear の公式を方程式(3-1)に代入すると，時刻を固定したときの非線形代数方程式が得られる．これらの方程式は，次のような反復法を用いて解かれる．

$$\boldsymbol{E}_k=\boldsymbol{g}(\boldsymbol{X}_k, t) \tag{3-5}$$

$$\begin{bmatrix} \boldsymbol{T} \\ \boldsymbol{F}(\boldsymbol{X}_k, \boldsymbol{E}_k) \end{bmatrix}\boldsymbol{X}_{k+1}=\begin{bmatrix} \boldsymbol{0} \\ \boldsymbol{B}(\boldsymbol{X}_k, \boldsymbol{E}_k) \end{bmatrix} \tag{3-6}$$

ここで，$\boldsymbol{X}_k=[\boldsymbol{I}_k, \boldsymbol{V}_k]$，$T$ は(3-1 a)と(3-1 b)から得られる N 行 $2N$ 列の係数行列，$\boldsymbol{F}=-(\partial \boldsymbol{f}/\partial \boldsymbol{X}), \boldsymbol{B}=\boldsymbol{f}(\boldsymbol{X}, \boldsymbol{E})-(\partial \boldsymbol{f}/\partial \boldsymbol{X})\cdot\boldsymbol{X}$ である．$\boldsymbol{F}, \boldsymbol{B}$ は(3-1 c)を $\boldsymbol{X}_k$ で線形化して得られる係数である．この偏微分係数を用いた線形化手順のため適当な初期値 $\boldsymbol{X}_0$ から出発した(3-5)，(3-6)の逐次代入過程は，Newton-Raphson 法なみの急速な収束性を持つことができる．

CIRCUIT では $\boldsymbol{X}$ の初期値を Gear の予測子公式(3-4)によって計算する．この予測子は一般に方程式(3-1)の非常によい近似解であるため，ほとんど反復過程は 2～3 回の反復で収束する．3 回の反復で収束しないときは，さらに反復代入をつづけるより，小さいステップ・サイズで新しい予測子を用いて再計算する方が効果的であることが開発過程で実験的に確認された．

方程式(3-6)の従属電流源の偏微分係数は，CIRCUIT トランスレータの前処理で必要最小限の項が拾い出される．すなわち電源要素を定義するすべての式の従属性を調べて，必要な微係数だけを取り出し，これらの微係数の計算手順を FORTRAN コードに翻訳し，目的モジュールに連結する．これらの前処理によって，偏微分を各反復ループで非常に効率よく計算することができる．この非零の偏微分係数の洗出しは，係数行列を疎に保ち，3.5 節で述べる疎行列操作の効果を十分発揮させるためにも重要な役割を果たしている．

過渡解析は 3 つの入れ子になったループからなる．もっとも内側のループは上に説明した非線形の代数方程式を解く反復過程であり，中間のループは打切り誤差を許容範囲におさめるように積分ステップ・サイズを調整しながら予測子と修正子を反復計算する数値積分の過程である．

もっとも外側のループは出力サンプリングを制御したり，解析終了条件をチェックするループである．

3.3 擬似過渡解析を用いた直流解

CIRCUIT においては，直流解を求めるのに過渡解析で用いたと同じ状態方程式 (3-1)

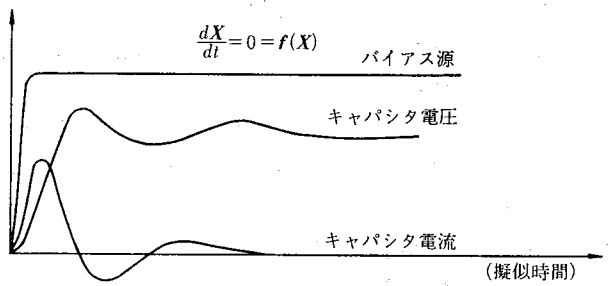


図 5　擬似過渡現象
Fig. 5　Pseudo transient

を利用する．この方法を擬似過渡解析と呼び，図5のように外部入力を一定としたときの過渡解析をすべての回路枝の電流電圧が整定するまで実行して，整定値を直流解とする．

図5に示すように，電圧の不均衡によって生ずる電流(微係数)が容量を充電し(積分し)，電位差を小さくする．最終的には電圧は均衡し定常解に到達する．この収束過程は実際の回路にスイッチを入れたときの現象と等価である．したがって，この方法によって求められた直流解は，非線形方程式の(一般的には)複数存在する解の中でもっとも現実的なものであるといえる．さらに，リアクタンスの積分過程が未知変数値の急峻な変化を緩和するから，非常に安定で通常の Newton-Raphson 法で見られるような出発値に依存する発散性は見られず，汎用の解析プログラムの使用する算法として適切であると思われる．

CIRCUIT では次のような工夫を加え，直流解析においてこの長所を有効に利用している．

1) 擬似過渡解析の緩和能力をリアクタンスを含まない純直流問題にも利用するために，擬似リアクタンスを内部的に付加して収束の安定性を保障している．図6に示すように，擬似キャパシタンスを非線形従属電流源と並列に付加し，擬似インダクタンスを非線形従属電圧源へ直列に付加する．定常状態では，擬似キャパシタンスを通るすべての電流と擬似インダクタンスでのすべての電圧降下は零になるので，これらの擬似リアクタンスは本来の回路の直流解に影響せず，一方において，反復計算の過程でこれらは非線形従属源に起因する電流ないし電圧の突変を吸収し，それを擬似過程の間に隣接する回路枝へ徐々に分配する働きをする．

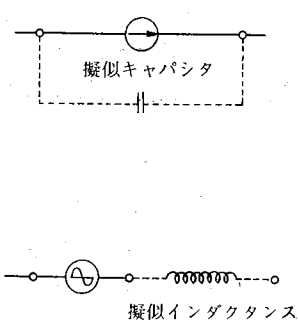


図 6　擬似リアクタンスの挿入
Fig. 6　Pseudo reactance substitution

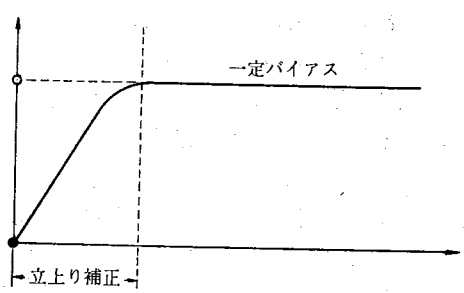


図 7　立上り補正
Fig. 7　Leading edge compensation

2) すべてのリアクタンス値はモデル記述での定義に関係なく同一の値に置き換えられる．これによって回路の時定数のばらつきをなくし，整定値に至るまでの積分ステップ数を少なくするように考慮している．この算法は広い範囲の時定数を持つ Stiff 系では非常に有効である．

3) すべてのバイアス電源に図7に示すような立上り補正を加える．この方法によって，直流解析開始時点で起きる電源の不連続な変化に対して微分器的に反応するリアクタンスがあるときでも，数値の発散が防止できる．このアプローチは抵抗のない非線形モデルに対して非常に有効である．

3.4 動作点まわりでの線形化と交流解析

非線形回路は動作点のまわりで線形化されてから，複素回路理論に基づいて交流解析が実行される．また動作点として，直流解を自動的に連結する機能も準備されている．CIRCUIT においては次の手順によって，回路方程式(3-1)を線形化する．

1) すべての非線形抵抗，コンダクタンス，キャパシタンス，インダクタンスを，動作点によって決定された値を持つ定数要素とする．

2) すべての独立電源は，交流解析用に定義された複素電源を除いて零に設定する．

3) 非線形従属電源は図8に示すように動作点のまわりの偏微分係数を値とする固定値の相互コンダクタンスとみなす．

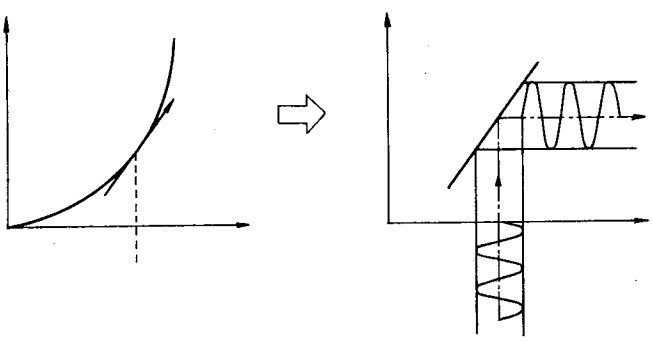


図 8 従属電源の線形化
Fig. 8 Linealization of dependent source

これらのすべての線形化手順は，直流・過渡解析のための方程式の構成過程に，陰に含まれるものである．したがって交流解析の基本方程式は，(3-1)のすべての変数を複素変数に変え，時間微分項 dx/dt を jwx で置き換え，動作点を係数行列に代入するだけで導くことができる．ここで，j は虚数単位，ω は角速度 $2\pi f$ (f は周波数)である．ソフトウェア的に重要なことは，行列内の非零要素の位置が直流・過渡解析の場合とまったく同じだということである．したがって直流・過渡解析と交流解析との両方に対して共通の疎行列操作手順を使うことができる．

3.5 擬似コード法による疎行列操作

上述のように，直流，過渡，交流解を求めるために，条件を変更しながら線形化した(3-6)を何度も反復して解かなければならない．そこで CIRCUIT の性能を向上するにはこの連立方程式の求解過程(パス)に要する計算時間を短縮することが非常に重要である．(3-6)を解くに当たって注目すべき重要な特徴が2つある．

1) 係数行列が非常に疎である．行列内の全要素に対する非零要素の比は，とくに大規模モデルでは，非常に小さい．

2) 行列内の非零要素の数および位置は1つのモデルの全解析中変化せず，その値だけが条件の変化に従って変わる．

Gustavson の算法はこのような行列の前処理を行って，もっとも効果的に分解を行うものである[3]．周知のように線形方程式の解を求める場合，Crout 法は次のように書ける．

$$\boldsymbol{AX}=\boldsymbol{B} \tag{3-7a}$$

$$(LU)\boldsymbol{X}=\boldsymbol{B} \quad (L/U\ 分解) \tag{3-7b}$$

$$U\boldsymbol{X}=L^{-1}\boldsymbol{B}=\boldsymbol{Y} \quad (前進代入) \tag{3-7c}$$

$$\boldsymbol{X}=U^{-1}\boldsymbol{Y} \quad (後退代入) \tag{3-7d}$$

各計算ステップは各要素間演算で書きくだすと，L の m 列と U の m 行は次の式で表現できる．

$$l_{im}=a_{im}-\sum_{u=1}^{m-1} l_{iu}u_{um} \qquad (i=m\sim N) \tag{3-8}$$

$$u_{mj}=\left(a_{mj}-\sum_{u=1}^{m-1} l_{mu}u_{uj}\right)\Big/ l_{mm} \qquad (j=m+1\sim N) \tag{3-9}$$

そして，ベクトル $\boldsymbol{Y}$ と $\boldsymbol{X}$ は次の式で計算される．

$$y_i=\left(b_i-\sum_{u=1}^{i-1} l_{iu}y_u\right)\Big/ l_{ii} \qquad (i=1\sim N) \tag{3-10}$$

$$x_i=y_i-\sum_{u=i+1}^{N} u_{iu}x_u \qquad (i=N\sim 1) \tag{3-11}$$

行列 L と V の全要素を求めるためには，N を係数行列の次元とすると $(N^3-N)/3$ 回の積和を計算しなければならない．したがって (3-8)～(3-11) をふつうに計算していたのでは，次元の大きな問題については膨大な計算時間を要して，現実的ではなくなってしまう．このような場合に行列の疎性を利用して効率よく計算を行う方法がいろいろと提案された．Gustavson の方法はそれらの中でもっと効率的であるが，非常に複雑な前処理を行う必要がある．したがって，行列を何度も反復して分解する場合に適している．もし疎行列の非零要素の位置があらかじめ解っていれば，(3-8)～(3-11)で定義されるすべての演算から非零要素間の算術演算のみをとりだすことができる．Gustavson の方法ではビット・マップを使って非零要素の演算をとりだし，図9に示すような形で FORTRAN の原始コードへ変換している．これらのコードはコンパイルされ目的モジュールへ連結されて，特定の問題のための専用の解法プログラムをつくりだす．

CIRCUIT では，この手順を UNIVAC シリーズ 1100 のハードウェアに適するように変更している．主な変更点をあげると，次のとおりである．

1) CIRCUIT の翻訳モジュールにおいて行列を内部的に表現するとき，ビット・マップの代わりに非零要素だけをとりだした添え字表 (packed index table) を使用している．

表 1　ビット・マップ法と添え字表方式の所要記憶域量
Table 1　Required memory area for bit map and packed index

要素数	ビット・マップ [語]	添え字表 [語]
200	1,113	1,200～ 2,000
300	2,501	1,800～ 3,000
500	6,946	3,000～ 5,000
1,000	27,778	6,000～10,000
2,000	111,113	12,000～20,000
3,000	250,001	18,000～30,000

```
      C(    238)=   -C(   116)*C(   124)-C(   148)*C(   159)
     *              -C(   170)*C(   180)-C(   190)*C(   199)
     *              -C(   208)*C(   216)-C(   224)*C(   231)+A(   108)
      C(    239)=   -C(   149)*C(   159)-C(   171)*C(   180)
     *              -C(   191)*C(   199)-C(   209)*C(   216)
     *              -C(   225)*C(   231)
      C(    240)=   -C(   117)*C(   124)-C(   150)*C(   159)
     *              -C(   172)*C(   180)-C(   192)*C(   199)
     *              -C(   210)*C(   216)-C(   226)*C(   231)
      C(    241)=   -C(   118)*C(   124)-C(   151)*C(   159)
     *              -C(   173)*C(   180)-C(   193)*C(   199)
     *              -C(   211)*C(   216)-C(   227)*C(   231)
      C(    242)=   -C(   119)*C(   124)-C(   152)*C(   159)
     *              -C(   174)*C(   180)-C(   194)*C(   199)
     *              -C(   212)*C(   216)-C(   228)*C(   231)
      C(    243)=   -C(   153)*C(   159)-C(   175)*C(   180)
     *              -C(   195)*C(   199)-C(   213)*C(   216)
     *              -C(   229)*C(   231)
      C(    244)=   -C(   120)*C(   124)-C(   154)*C(   159)
     *              -C(   176)*C(   180)-C(   196)*C(   199)
     *              -C(   214)*C(   216)-C(   230)*C(   231)
      Y(     36)= (-C(    63)*Y(    14)-C(   116)*Y(    25)
     *              -C(   138)*Y(    23)-C(   148)*Y(    31)
     *              -C(   170)*Y(    32)-C(   190)*Y(    33)
     *              -C(   208)*Y(    34)-C(   224)*Y(    35)
     *              +B(    36))/C(   238)
      C(    245)= (-C(   148)*C(   160)-C(   170)*C(   181)
     *              -C(   190)*C(   200)-C(   208)*C(   217)
     *              -C(   224)*C(   232))/C(   238)
      C(    246)= (-C(   148)*C(   161)-C(   170)*C(   182)
     *              -C(   190)*C(   201)-C(   208)*C(   218)
     *              -C(   224)*C(   233))/C(   238)
      C(    247)= (-C(   116)*C(   125)-C(   148)*C(   162)
     *              -C(   170)*C(   183)-C(   190)*C(   202)
     *              -C(   203)*C(   219)-C(   224)*C(   234))/C(   238)
      C(    248)= (-C(   116)*C(   126)-C(   148)*C(   163)
     *              -C(   170)*C(   184)-C(   190)*C(   203)
     *              -C(   208)*C(   220)-C(   224)*C(   235))/C(   238)
      C(    249)= (-C(   148)*C(   164)-C(   170)*C(   185)
     *              -C(   190)*C(   204)-C(   208)*C(   221)
     *              -C(   224)*C(   236))/C(   238)
      C(    250)= (-C(   116)*C(   127)-C(   148)*C(   165)
     *              -C(   170)*C(   186)-C(   190)*C(   205)
     *              -C(   208)*C(   222)-C(   224)*C(   237))/C(   238)
      C(    251)=  A(   109)
      Y(     37)= (-C(    68)*Y(    15)-C(    94)*Y(    23)
     *              +B(    37))/C(   251)
      C(    252)=  A(   110)
      Y(     38)= (-C(    71)*Y(    16)-C(    95)*Y(    23)
     *              +B(    38))/C(   252)
      C(    253)=  -C(   101)*C(   109)-C(   149)*C(   160)
     *              -C(   171)*C(   181)-C(   191)*C(   200)
     *              -C(   209)*C(   217)-C(   225)*C(   232)
     *              -C(   239)*C(   245)+A(   111)
      C(    254)=  -C(   102)*C(   109)-C(   150)*C(   160)
     *              -C(   172)*C(   181)-C(   192)*C(   200)
     *              -C(   210)*C(   217)-C(   226)*C(   232)
```

図 9 Gustavson の方法によって生成された FORTRAN コードの例
Fig. 9 Picked up operations by Gustavson's algorithm

この方法によれば，大規模モデルではビット・マップ法よりはるかに少ない記憶領域ですむ．表 1 は上記 2 つの方法を比較した例であるが，添え字表方式の優位性を示している．

2) FORTRAN コード生成の代わりに，擬似コード生成法を選択した．これらの 2 方法の比較を表 2 に示す．

擬似コードは算術演算のコードとオペランドの添え字によって構成される可変長の

表 2 FORTRAN コード法と擬似コード法の比較
Table 2 Comparison between FORTRAN code and pseudo code

	FORTRAN コード	擬似コード
実行速度	1.0	1.2～1.5
記憶域	10.0～70.0	1.0
オーバヘッド	コンパイラとプログラムのオーバレイ	0

データ列である．これらのコードは各パスごとに解釈されながら実行される．比較表は，計算時間と所要記憶容量を総合すると FORTRAN コード法より擬似コード法が優れていることを示している．UNIVAC シリーズ 1100 を用いての評価によれば小規模モデルの解析においても，擬似コード法の所有解析時間は FORTRAN コード法より小さかった。これはコンパイルに要する時間が節約できるためである．

以上，2つの大きな変更点の他に，CIRCUIT では以下の修正を Gustavson の方法に加えて，解析に要する記憶容量と計算時間の短縮を図っている．

すなわち，Crout 分解前と分解後の行列を同一の領域に記憶できるように，非零行列要素の添え字のつけ方に工夫を加えた．さらに選び出した非零要素間演算を，1回の解析について一度計算すればよい部分と，パスごとに繰り返し実行すべき部分に分離し，パスに要する計算時間を短縮する努力をしている．また，(3-6) の並び順は，Crout 分解後の行列の fill-in 要素 (分解前に零であった位置に生ずる非零要素) の数に密接に関係している．CIRCUIT では文献[1]に紹介されている Markowiz の重み関数 $W_{ij}=(R_i-1)(C_j-1)$ を利用して，行と列を同じ順序で入れ替えるための重み関数

$$W_i=(R_i-1)(C_i-1)$$

を導き，fill-in 要素の発生がなるべく少ない方程式と未知数の並べ順を決定している．ここで R_i は i 行目の C_i は i 列目のそれぞれ非零要素の数である．W_i は，Crout 分解の過程で第 i 行，および第 i 列に発生する fill-in の要素数の上限を意味する量で，CIRCUIT では W_i の昇順に方程式 (3-6) の行と列を入れ替えている．

表3は，UNIVAC 1100 を用い，次の3つの場合について実行列を示している．例1は直列・過渡解析を 150 回繰り返す統計解析の結果であり，例2のモデルは例1のモデルを10個並べて構成した回路であり，例3のモデルは例1のモデルを20個並列に並べて作成したものである．それぞれ通常の直列・過渡解析を行った場合のデータで，パス数はいずれも 828 回であった．

表 3 UNIVAC シリーズ 1100/82 での実行データ
Table 3 Performance data on UNIVAC 1100/82

例	素子数	パス回数	前処理 [秒]	解析 [秒]	所有記憶量 [キロ語]
1	53	105, 100	27	1, 840	38
2	532	850	245	145	91
3	1, 041	850	632	290	150

例1のデータは前処理のオーバヘッドが統計解析あるいは最適化解析では無視できるほど小さくなり，疎行列処理が非常に有効であることを示している．例2と例3は，オーバヘッド時間は大きいけれども総計算時間が十分実用に足るため，本節で述べた前処理が依然として有効であることを示している．

4. おわりに

CIRCUIT の機能を支えている基本的な数値計算法について，その概略と採用の効果について述べた．これらの手法をアプリケーション・プログラムとしてまとめあげるために，ソフトウェア技術面からの考察と，プログラミング上の工夫が必要であったことも付記しておきたい．

CIRCUIT は，発表以来多くのユーザからいただいた貴重な助言をもとに，次のような

機能の追加や効率の改良を行っている.

・交流解析における群遅延特性の計算機能

・ループ形のパラメタ変更機能と，直流伝達特性のプロット機能

・ケース・スタディの結果の同一画面上へのプロット機能

・状態変数の並び順の工夫による計算時間の短縮

また，現在作業中あるいは，計画中で，近々追加される予定の機能としては

・LOAD プロセッサを利用した，小規模モデルの簡易トランスレーション機能

・節点電位による，従属性の記述と，初期条件の設定機能

・計算結果をファイルを通じて，他のタスクへ引き継ぐ機能

・所用計算時間の短縮

などがあげられる.

最後に，CIRCUIT の開発をともにし，本稿にも大きな役割を果たした，日本ユニバック(株)の宮田豊雄，山内正史の両氏に感謝したい.

参考文献 [1] W. T. Weeks and A. J. Jimenez, et al., "Algorithms for ASTAP—A Network Analysis Program," *IEEE Trans. on Circuit Theory*, Vol. CT-20, No. 6, Nov. 1973, pp 628-634.

[2] R. Brayton, et al., "A New Efficient Algorithm for Solving Differential-Algebraic System Using Implicit Backward Differentiation Formulas," *Proc. IEEE*, Vol. 60, No. 1, Jan. 1972, pp. 98-108.

[3] T. G. Gustavson, et al., "Symbolic Generation of An Optimal Crout Algorithm for Sparse Systems of Linear Equations," *J. ACM*, Vol. 17, No. 1, Jan. 1970, pp. 87-109.

執筆者紹介 長島 毅 (Takeshi Nagashima)

昭和22年生，45年東京大学工学部航空学科卒業．47年同大学院修士課程修了．同年日本ユニバック(株)入社．以来主として，機械・制御・電気電子などの工学分野における計算機シミュレーション技術の応用に関する業務に従事．連続系シミュレータ CSSL 1100 の導入・改良，電子回路シミュレータ CIRCUIT の設計・開発，論理回路シミュレータ DIANA の設計・開発などのプロジェクトに従事．現在，応用ソフトウェア部技術計算グループに所属.

米国国防総省開発言語 Ada の応用分野

1. 埋込み型システム記述用言語

米国国防総省(Department of Defence—DoDと略す)が埋込み型コンピュータ・システム(embedded computer system)記述用の言語を統一する目的で制定を急いでいる言語 Ada については，その開発経緯[1],[2],[3]や言語仕様の概要[4]についての紹介記事が雑誌に掲載され，かなり国内でも知られるようになってきた．

DoD 自身には Ada を DoD 専用の言語とする考えはなく，広く一般にも Ada が普及することを望んでいる．Ada が DoD に納入されるすべてのコンピュータで使用可能になるとすれば，遠からず同じコンピュータを使用する民間ユーザでも Ada が使用可能になるということが予想される．

そこで，そのような時代に備えて一般のユーザが Ada をどんな応用分野でどう利用していくべきかについて，発表されている言語仕様[5],[6]をもとにして考察しておきたい．

2. Ada 本来の応用分野

Ada はもともと埋込み型コンピュータ・システムを主な応用分野としている．埋込み型コンピュータ・システムは，コンピュータのハードウェアと一体化させて，レーダ管制システム，ミサイルや人工衛星の指揮系統システム，通信制御システムなどに使われるソフトウェア・システム一般を指している．

この目的のために Ada がとくに備えている機能としては

・データをリアルタイムに処理する同時並行操作
・特殊な入出力機器対象の低水準入出力

をあげることができる．これ以外の各種の機能はとくに埋込み型コンピュータ・システム用というよりも，ソフトウェア工学，構造化プログラミングを実現するために信頼性，保守性，効率性，単純性，実現性，機械独立性を高めるべく用意されているもので，一般のプログラム言語にも求められるものである．

この本来の目的に沿った使い方をするとき，Ada の本領がもっとも発揮できるはずである．この分野としてあげられるのは，自衛隊関係の防衛システムのみならず，

・平和目的の人工衛星やロケットの制御システム
・道路交通管制システム
・航空管制システム
・列車運行管理システム
・気象用レーダ管制システム
・工場の制御用コンピュータ・システム

などがあげられよう．この方面については，今後 Ada の活用が期待できる．

3. 事務計算

事務計算分野でとくに必要とされる機能は，

・ファイル処理
・10進固定小数点演算
・帳表用編集機能
・整列併合機能
・表操作

であろう．Ada についてこの点を考えてみると，表操作は配列とレコードで，整列併合はサブルーチンの呼出しでそれぞれ十分対応できるものの，その他の機能については能力不足であるといえる．

まずファイル処理であるが，Ada はユーザ向きの入出力としては順ファイル処理とテキスト・ファイル処理だけを備えている．キーを指定してコードを処理する索引ファイルや相対ファイルの機能は別途パッケージを用意する必要があり，入出力のような基本的な操作を実引き数を指定するサブルーチン呼出しで行うのはかなり面倒であろう．

10進固定小数点演算は1円の誤差も出したくない事務計算には必須の演算である．Ada は固定小数点データを備えているが，これは COBOL や PL/I の10進固定小数点と少し異なっている．COBOL や PL/I では，10進法に限った場合に誤差を出さない加減乗算を可能にしているが，Ada では単に誤差を浮動小数点演算のような相対誤差ではなく，絶対誤差にとどめるというにすぎない．実現方法は機種に依存して決まるのであろうが，文献[5],[6]を見る限り，2進固定小数点演算で任意進法の固定小数点演算をシミュレートさせることを意図している．この

方法では，小さいといえども誤差があり，事務計算が必要としている目的に不十分と思われる．

帳表用編集機能は COBOL の報告書作成機能とピクチャによる数値の文字表現の両方を意味しているが，Ada はこのどちらも備えていない．わずかに PASCAL 的なテキスト入出力を備えているのみであり，これはたとえば 1230 という数値を 1,230 と編集したい場合には，不便であろう．編集用のパッケージを用意すればよいのであろうが，これも COBOL や PL/I に比べてわずらわしい．

以上の理由により，事務計算分野での Ada の使用にはかなりの難点がある．同じ理由により，埋込み型コンピュータ・システムであっても，銀行のオンライン・システムのような事務用のシステムには向かない．

4. 技術計算

技術計算分野でとくに必要とされる機能は

・浮動小数点演算　　・配列

・数学的組込み関数　　・コード生成の最適化

である．Ada は浮動小数点演算と配列に関しては，十分な機能を備えている．コード生成の最適化については，設計時にかなり意識されており期待できる．

ところが，SIN, COS, LOG などの数学的組込み関数は一切備えていない．このことは一見 Ada が技術計算には向かないように見える要因である．しかし，埋込み型コンピュータ・システムでも三角関数などは必要であることを考えると，Ada がパッケージ能力に優れていることに思いいたる．すなわち，数学的組込み関数をパッケージとして用意しておけば，使い方も関数引用という点で FORTRAN と変わるところはない．統計パッケージ，グラフィック・パッケージの利用も同様にできよう．

このようにみると，Ada が技術計算に向いていないという積極的な理由はまったく見当たらない．わずかに書式の多様さで劣る程度である．したがって，現在 FORTRAN あるいは PL/I がカバーしている技術計算分野にはかなり Ada が進出してもよさそうに思える．

5. システム記述

システム記述分野で要求される機能は，

・ビット処理　　・抽象データ構造

などであろう．Ada のコンパイラは，Ada 自身で記述することが期待されている．PASCAL コンパイラが PASCAL で記述できることを考えると，PASCAL の大規模な拡張ともいえる Ada はこの方面の能力を十分備えていると考えられる．

ただし，Ada の設計方針としてそもそもオペレーティング・システムを記述する程度の細かな配慮はされていないので，CONCURRENT PASCAL や言語Cの代替えとしては期待すべきでない．

6. 教育用

Ada は新しい概念を多く備え，機能が豊富で複雑な言語である．他言語を知っていても，習得にはかなりの困難がともなう．しかし，PASCAL に似たスマートな言語でもあるので，適切な入門書とサブセットで教えれば，PASCLA 同様，プログラミング教育用の言語としても十分使うことができる．

7. Ada の応用分野

以上，言語 Ada の応用分野について，その機能面から向き不向きを略述した．まとめると

1) 向いている応用分野
   - 埋込み型コンピュータ・システム（事務用は除く）
   - 一般の技術計算
   - 言語プロセッサなどのシステム記述
   - プログラミング教育用
2) 向かない応用分野
   - ファイル，帳表を重視する事務計算
   - オペレーティング・システムの記述

Ada の言語仕様はまだ最終決定されているわけではなく，今後の改訂によって応用分野も変化する可能性がある．Ada は複雑な面を持っているが，COBOL, FORTRAN, PL/I にはないよい面も多数持っており，このよい面を生かして今後のソフトウェア工学の発展に貢献させていくべきと考える．

参考文献

[1] 大野恂郎，“Ada—米国国防総省の新言語”，*bit*, Vol. 11, 7月号, 1979.

[2] 真田正二，“米国防省の主流となるか？ 新言語 Ada の素顔”，学習コンピュータ，5月号，1980.

[3] 日本電子工業振興協会，“米国国防総省HOLプロジェクト”，ソフトウェア・エンジニアリングに関する調査，pp 230-234.

[4] TSINKY，“新しいプログラミング言語 ADA”，*bit*, Vol. 12, 3月号, 1980.

[5] “PRELIMINARY ADA REFERENCE MANUAL”，*SIGPLAN NOTICES*, Vol. 14, No. 6, June 1979, PART A.

[6] “Rationale for the Design of the ADA Programing Language”，*SIGPLAN NOTICES*, Vol. 14, No. 6, June 1979, PART B.

[7] United States Department of Defence，“プログラム言語 Ada の基準文法書(英文)”，*bit* 別冊，1980.

（真田　正二，1100 ソフトウェア開発部）

DDX について

1. データ通信

日本で本格的なデータ通信システムが出現したのは東京オリンピックの開催された昭和39年といわれている．以後データ通信は急速に普及し，データ量の少ないシステムや地域分散による遠距離ネットワークが急増している．さらに最近はインテリジェント・ターミナル，コミュニケーション・プロセッサなどの出現により通信処理機能の分散化，コンピュータ・ネットワークの構築が行われている．この結果，ホストコンピュータの負荷軽減，リソースの共用，システムの融通性，経済性，信頼性の向上が図られるようになった．このようなデータ通信の変革にともない，従来の通信回線は種々の技術的な限界に直面している．これは従来の通信回線が音声伝送用に設計された設備を使用しているためで，たとえば伝送帯域は 300～3,400 Hz に制限されている．このためディジタル伝送ができず，モデムを使用したアナログ伝送をしなければならない．また通信速度も最高で 9,600BPS 程度に押えられる．さらに交換回線においては交換機が介在するため，音声伝送では問題にならないインパルス性雑音等によりデータ品質が悪化したり，ダイヤルしてから接続が完了するまでにかなり時間がかかる（最大約15秒）などの問題があり，かなり支障をきたしている．このような通信回線の技術的な制約を取り除くため，世界各国で半導体技術とディジタル伝送技術をとりいれた高品質で信頼性の高い経済的な公衆データ網（PDN: Public Data Network）の開発が進められている．日本においても昭和54年の12月から日本電信電話公社（以下，電電公社と略す）により PDNである“新データ網サービス”通称 DDX（Digital Data Exchange）の提供が開始された．

2. DDX の種類

DDX には回線交換サービスとパケット交換サービスの２種類がある．回線交換サービスは任意の相手と高速，高品質の通信ができる交換網であり，現在，東京，横浜，名古屋，大阪の４都市でサービスが提供されている．パケット交換はデータをパケットと呼ばれる小ブロックに区切って交換機がこれを蓄積しながら伝送するもので伝送品質が優れており，さらに異速度の相手とも通信できるサービスである．パケット交換サービスは現在，東京，横浜，名古屋，大阪，福岡，札幌，仙台でサービスが提供されている．

3. 回線交換サービス

回線交換サービスは，従来の公衆通信回線を高速化，高品質化したものといえる．従来と異なる点は回線を流れるデータがすべてディジタル信号であることである．このため交換機は時分割ディジタル交換機が使われ，伝送路も PCM 方式の全二重ディジタル伝送路が使われる．データは時分割多重化された形で伝送路を通り，交換機で交換される．これを図１に示す．

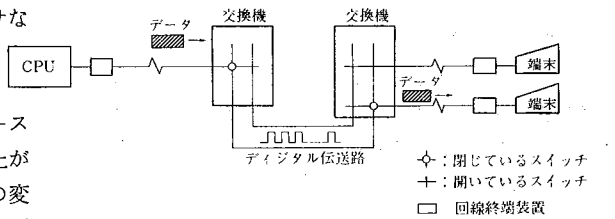


図 1　回線交換網の原理

3.1 回線交換サービスの特徴

回線交換サービスはディジタル伝送路，時分割電子交換機が使われていることから，次のような特徴を持っている．

（1） 接続時間の短縮

回線交換では相手との接続動作をキャラクタ・ダイヤルにより行うことができる．キャラクタ・ダイヤルとは図２に示すように相手番号を，たとえばコンピュータのバッファに JIS７単位符号の数字でセットしデータとして，交換機に送出するものである．2,400 BPS の場合８桁のダイヤルを行うのに要する時間は約0.03秒で，相手に呼出しがかかるまでの時間を加えても 0.5～１秒程度ですみ，従来の公衆通信通信回線に比べれば，時間が１桁程度短縮できる．

（2） データ品質，通信速度の向上

回線交換サービスに使われる電子交換機はスイッチ部も半導化されており，雑音がきわめて少ない．伝送路もディジタル化されており，雑音や歪に強い．このためデータ品質は従来の回線に比べて，ビット誤り率で１桁以上向上している．また PCM 伝

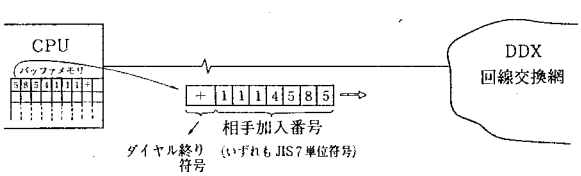


図 2　キャラクタ・ダイヤル

送路は電話の音声1回線分を64キロビットのデータを使用して伝送しており，これをデータ通信で使えば電話1回線で 48,000BPS の通信ができる．

（3） 各種付加サービスの提供

交換機に蓄積プログラム制御電子交換機が使われているため既存網にない各種付加サービスが提供されている．これには接続相手を限定する“閉域接続”，発呼表示をするだけであらかじめ定められた相手に接続する“ダイレクトコール”，接続が完了したとき発信側着信側それぞれに相手の加入番号を送ってくれる“相手通知サービス”，通信料をコンピュータ・センタなど1か所に課金させる“料金の一括払い”などがある．

3.2 回線交換サービスの加入者インタフェース

加入者インタフェースにはXインタフェースとVインタフェースの2種類があり，図3に示す4つの加入形態がある．図3の①はXインタフェースの場合で加入者宅内に DCE (回線終端装置) が設置される．Xインタフェースには調歩式の X.20 と同期式の X.21 がある．Xインタフェースは従来のものに比べ信号線の数が減少し，電気的特性も従来のトランジスタを想定したものから IC の特性に見合ったものになっており，合理的なインタフェースといえる．X.20 による接続制御の方式を図4に示す．図3の②は従来のVインタフェースの場合で，利用者宅内に DCE と NCU (網制御装置) が設置される．

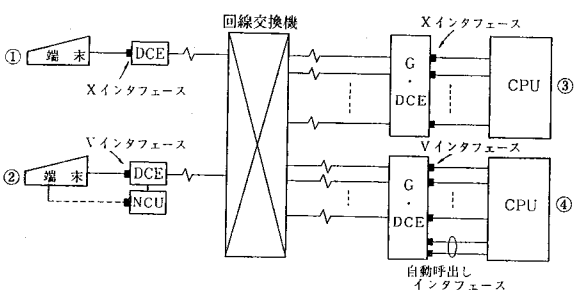


図 3 回線交換サービスの加入形態

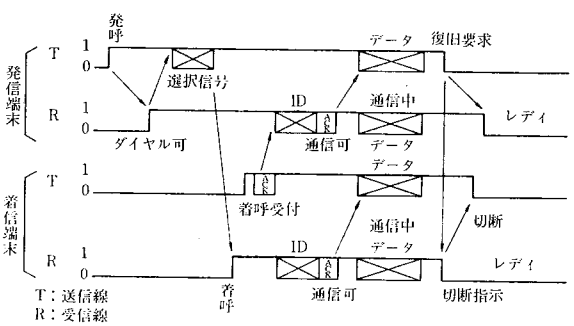


図 4 X. 20 接続制御シーケンス

相手との接続は NCU を使い，手動または自動で行われる．図3の③は，①と同じであるが，DCE が多回線用となったものである．④はVインタフェースの多回線用で，DCE は自動発信タイプの NCU を2セット内蔵している．

3.3 課金方式および適用業務

回線交換の料金は基本料と回線保留時間による通信料から構成される．したがって回線交換は短時間に大量のデータが高密度に発生するリモート・ジョブ・エントリ，データ・ギャザリング・システムなどに向いているといえる．

4. パケット交換

パケット交換は伝送するデータを 128, 256 文字などのブロックに分割し，さらに宛先・制御情報をつけたパケットと呼ばれるブロックに組み立てて送受信する方式である(図5)．パケットとは荷札をつけた小包という意味である．端末機から交換局に送信されたパケットはいったん蓄積され，宛先・制御情報が読まれ該当する相手局に局間転送される．パケットを受信した相手局でも同様に宛先・制御情報を見て，それが自局に収容されている端末機宛のものであれば該当端末機に配信する(図6)．このようにパケット交換ではデータを蓄積しながら通信が行われる．

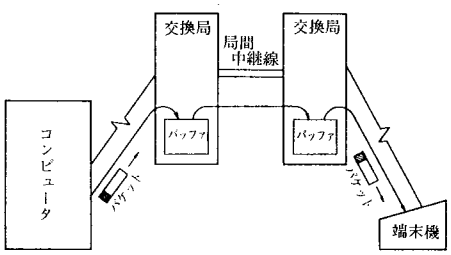


図 6 パケット交換の原理

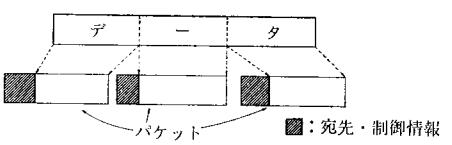


図 5 データのパケット化

4.1 パケット交換の特徴

パケット交換はデータの蓄積を行うため，次のような長所を持っている．

1) 異速度端末機間の通信が可能
2) 伝送制御方式の異なる端末機間の通信が可能
3) 局間中継線はパケットが流れるときしか占有されないので使用効率がよい（ビット当たり単価が安い）．

4) 回線保留時間が課金の対象にならないで実際に伝送したデータ量に見合った合理的な課金が行われる.

反面，パケット交換の短所としては

1) いったん蓄積するため伝送遅延が発生する.
2) パケット分解組立て機能が必要である.
3) データ転送フェーズにおいても交換局が介入するため，手順が制約される.

などがあり，システム設計上，これらの利害得失を十分考慮する必要がある.

4.2 端末機の種類および加入方法

パケット交換に加入する端末機はパケット形態端末機 (PT) と一般端末機 (NPT) に分けられる. PT は自分でパケット分解組立て機能を有するもので，通常はコンピュータ，インテリジェント・ターミナル等である. 特徴として1本の物理的回線に最大4,096 までの論理チャネルが設定ができる，これをパケット多重通信という. NPT はパケットの分解組立てを網内のパケット分解組立て機能 (PAD) に依存するものである. これらを図7に示す.

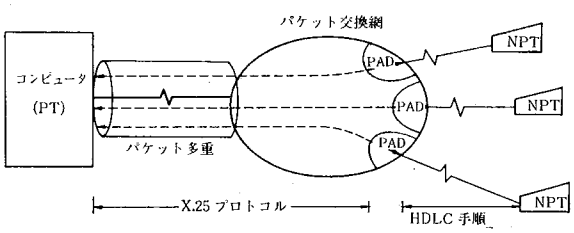


図 7　PT と NPT および各手順

一方，パケット交換の場合も回線交換同様VおよびXインタフェースで加入できる. また接続の形態としては相手固定接続 (PVC) と相手選択接続 (VC) がある. 網との制御手順には PT に対して X.25 プロトコル，NPT に対しては HDLC 手順，ベーシック手順，デリミタ手順の3つがある.

4.3 課金方式および適用業務

料金は基本料と伝送パケット数による通信料から構成され，回線保留時間は課金の対象にならない. したがって，パケット交換は TSS 等データ密度が低く回線保留時間の長いシステムに向いている.

また，先の特徴を生かし，今後はコンピュータ・ネットワーク，災害対策を考慮した地域分散システム等の分散処理システム構想に幅広く適用されてゆくことになろう.

参考文献

[1] 電気通信協会編，データ交換の基礎知識，オーム社，1978.
[2] 日本電信電話公社編，回線交換サービスのインタフェース（技術参考資料），日本データ通信協会，1980.

（山浦　史雄，データ・コミュニケーション部）

FTCS について

1. フォルト・トレラント技術の現状

従来，フォルト・トレラント技術は宇宙航空，軍事用，あるいは交換機のきわめて特殊な分野においてのみ研究され，実現化されてきた. しかも，そのほとんどが冗長構成を用いるものであった. しかしその後のハードウェア，ソフトウェア技術の発達，とくに半導体技術の目覚しい発展により，商用システムにも，この技術の適用が可能となりつつある.

また，フォルト・トレラント技術では階層化の傾向がある. これまで1つの独立したシステム（コンピュータ）の範囲で考えてきたものが，いまや半導体チップのレベルから考慮する必要が生まれた. さらにハイエンドとしては広域ネットワーク・アプリケーションにおけるフォルト・トレランスを考える必要が生じてきた.

さらには，フォルト・トレランスを実現する上でソフトウェアの役割が重視されてきたことである. この傾向は一層強まり，単なるシステム・リソースを活用するソフトウェア技術のみならずソフトウェア自身の障害をも考慮したフォルト・トレラント技術が要求されつつある.

2. FTCS

フォルト・トレラント技術を論ずる場としては FTCS (Fault Tolerant Computing Symposium) がある. この会議は IEEE Computer Society が主催しており，年1回，米国を中心に開催されており，今年で10回目を迎える. FTCS は IEEF-CS の中の TIC-2 (Technical Interest Council) のソフトウェアとアプリケーション部門の10の TC (Technical Committee) の1つであり，超高信頼化システムを実現するためのソフトウェアおよびハードウェア技術を取り扱うものである.

55年度 (FTCS-10) は去る10月1日より3日間，京都会館にて開催された.

シンポジウムは「これからますます拡大する情報化社会において，コンピュータの故障は社会に対し重大な損害 (serious damage) を与える. したがって，フォルト・トレラント技術の重要性はますます

増大する」という大島信太郎氏のオープニングにより開始され，21のセッションが2つの会場に分かれて行われた．

全体的に見た今年の特徴は，昨年同様に Testing/System Diagnosis の分野であり，18の論文が発表された．そのうち4編は，マイクロプロセッサと PLA のテストに関するものである．自動検査法もまた重要なテーマで7編の論文が発表された．テスト法について目立ったものは，システム信頼性とパフォーマンスの評価に関するものであり，14編の論文が発表された．その他，大容量記憶やメモリにおけるコーディング技術に関するもの，設計の検証に関するものが目立った．ソフトウェアの信頼性技術に関する論文が，昨年以来急に増加しているのも大きな特徴といえる．

3. いくつかの論文の概要

少ない紙面のため，ここでは Sperry Univac 社の技術者により発表された論文を中心に紹介する．

まず第1日目，L. A. Boone が発表した "Availability, Reliability and Maintainabiliy Aspects of the Sperry Univac 1100/60"(他に H. L. Liebergot, R. M. Sedmak が共同執筆者) がある．UNIVAC 1100/60 は一昨年の NCC で発表され，今年から客先設置が行われている最新システムであるだけに，出席者の関心も高かった．論文内容は UNIVAC 1100/60のARM設計思想について述べ，ARMを構成する基本機能である障害の発見(fault detection)，エラー補正 (error correction)，障害の分離 (fault isolation)，エラー回復(error recovery)，フォルト・インジェクションおよびメンテナンスのそれぞれについて詳細な説明がされた．説明の中でもインストラクションの自動再試行と SSP (system support processor) を用いて行う多重プロセッサ・システムにおける移植手法による二段階のエラー回復の考え方は，他社に見られないユニークなものであり，多くの人の関心を引いた．メンテナンスのための各種の機能についても，その重要性を強調していたが，FTCS としては印象的であった．なお，同じセッション (Commercial Systems) の中では，この他に Amdahl 社と日本電気(株)からそれぞれ 470V/6 および ACOS 900 システムのフォルト・トレラント技術の紹介がなされた．

従来 1,600 BPI あるいは 6,250 BPI の磁気テープにおいては，主要なエラー・モードは単一トラックのバースト・エラーであったので，単一トラック・エラー修正 (STEC) コードが採用されてきた．しかし，6,250 BPI を越える磁気テープが現れて来ると，これに対応できるエラー修正コードの開発が要求される．A. K. Bhatt と L. L. Kinney による "Random-Double-Track-Error Correction in Magnetic Tapes (RDTEC)" では，Reed-Solomon コードを応用したランダムのダブル・トラック・エラーの修正を可能にする RDTEC を提案し，9トラックおよび10トラックの例について説明を行っている．この他にバブル・メモリや CCD のようなシフト・レジスタ型メモリのエラー・コード等の論文も紹介された．

D. R. Mueller, D. J. Kunshier の "Support Processor Based System Fault Recovery"は，System Level Diagnosis のセッションで発表された．当論文は UNIVAC 1100/10 の時代に始まる支援プロセッサ (SP) 出現の背景を述べ，UNIVAC 1100/80, 1100/60 を経て完成された SP の技術を紹介している．SP の特徴はシステム制御機能を集中化している点であり，そうすることによりシステム制御機能を明確に体系づけて，標準化を実現している点であろう．SP はきわめて広範な機能を持っているが，論文ではフォルト・トレランスの観点から制御記憶装置のエラー修正，移植ダイナミック・リコンフィギュレィションについてとくに言及している．中でも関心を引いたのは，こうしたフォルト・トレラント機能を SP に持たせたことにより，どの程度エラー回復能力が向上したかを数量的に評価している点である．SP はこれからも ARM 機能を重視する上で中心的な役目を果たすものといえる．

(伊東　玄，プロダクト・サポート統括第1部)

BOOKS

——ZOHAR MANNA 著——

"Lectures on the Logic of Computer Programming"

CBMS-NSF Regional Conference Series in Applied Mathematics 31

SIAM, A 5変形判, iv+49 pp. 1980

SIAM (Society for Industrial and Applied Mathematics) では，米国の CBMS (Conference Board of the Mathematical Sciences) の指示により，NSF (National Science Foundation) の後援を受けて応用数学の分野におけるホットな研究テーマについて一連のレクチャ・ノートの刊行を続けている．本書はこのシリーズの31番目に当たる．他に比較的最近に発行されたものに，F. S. Roberts, "Graph Theory and Its Applications to Problems of Society", J. Hartmanis, "Feasible Computations and Provable Complexity Properties", E. L. Johnson, "Integer Programming", S. Winograd, "Arithmetic Complexity of Computations"などがある．

本書は Zohar Manna 教授が Rensselaer Polytechnic Insititute で行った講義のノートであり，わずか49ページの小冊子である．

著者の Zohar Manna 教授は Stanford 大学の J. McCarthy 教授[1]や R. W. Floyd 教授[2]のアイディアを忠実に継承して，プログラミングの諸問題を記号論理を応用して研究してきており，現在では自他ともに許す「プログラミングの理論」の第一人者である．同教授の著作には，現在標準的な参考書と考えられる "Mathematical Theory of Computation (McGraw-Hill, 1974)" があり，わが国のこの分野の第一人者である現筑波大学の五十嵐滋教授によって "プログラムの理論（日本コンピュータ協会，1975)" として既に紹介されている．したがって，Zohar Manna 教授についてはよく知られていることと思う．

内容は，理論を説くというよりはプログラミングにおける実践的な技法を紹介しており，主要なプログラムの正当性と停止性の検証方法や，プログラムの不変表明の組織的注記方法，さらに仕様から変換法則によってプログラムを合成する方法を，具体例を用いて説明している．平明な説き方は，1978年の Z. Manna and R. Waldinger, "The Logic of Computer Programming", IEEE Transactions on Software Engineering, Vol. SE-4, No. 3 とよく似ている．内容的には，本書では，recursive program や Scott の fixed-point semantics は，はぶかれており，その代わりに，Z. Manna and R. Waldinger, "Synthesis: dream⇒programs", IEEE Transactions on Software Engineering, Vol. SE-5 に沿うプログラム合成が加えられている．このように，本書は，わずか49ページの中に，斯界の第一人者がプログラミングにおける検証，合成の技法を概説した小冊子であり，むしろプログラマに一読をすすめたい．値段は SIAM 会員には 6 ドル，非会員には 7.5ドルである．

さて，本書は6章からなり，とくに1章から4章までがプログラマにすすめられる．各章の内容を見てみよう．

＜1章　Partial Correctness＞

Specification, partial(あるいは total)correctness, termination などの概念を説明してから，Floyd-Hoare の invariant 法と，Morris-Wegbreit の subgoal 法を簡単な具体例を使って説明している．また，この2つの方法による correctness proof が互いに effective に変換できるという意味で同等の能力を持つことを示している．

＜2章　Termination＞

停止性の証明法として，well-founded set と termination function を導入して，その使い方を例示している．また Dershowitz-Manna の multi-set ordering による証明法を紹介している．

＜3章　Total Correctness＞

Z. Manna and R. Waldinger, "Is 'sometime' sometimes better than 'always'?", Comm. ACM (1978)によって色々と議論[3]を生んだintermittent 法の紹介をしている．また，termination を仮定した invariant proof と subgoal proof, および well-founded ordering proof は effective に intermittent proof に変換できるという意味で，intermittent

法がそれらの方法と同等の能力を持つ点を強調している.

＜4章 Systematic Program Annotation＞

invariant 法によってプログラムの correctness proof を行うためには，プログラムの途中の invariant assertion を注記しなければならない。invariant assertion の(半)自動生成はいろいろと研究されているが，ここでは Dershowitz-Manna の方法を解説している．元来，プログラムの注記はプログラミングでふつうに行うものであるから，この方法の枠組は自動化に頼らずとも実用化できる.

＜5章 Synthesis of Programs＞

weakest precondition とその変換法則ないし原理によるいわば semantical な合成のアプローチを，notation は多少異なるが，"synthesis: dream⇒programs" の線に沿って解説している。さらに，いわゆる構造化プログラミングによるアプローチとこの合成によるアプローチを，プログラム作成例によって比較している.

＜6章 Termination of Production Systems＞

微分などの記号処理のいわゆる Markov-Post 流のプロダクション系の停止性の証明を，nested multiset, regular, multiset proof によって紹介している.

すでに述べたように，この小冊子は理論家よりはむしろ実務家向に概念と技法の意味内容を概説するものであり，49ページの分量であることからも，多忙な実務家向きではないだろうか.

引用文献

[1] J. McCarthy, "A Basis for a Mathematical Theory of Computation", P. Braffort & D. Hirschberg eds., *Computer Programming and Formal Systems*, 2nd Printing, North Holland, 1967.

[2] R. W. Floyd, "Assigning meanings to programs", *Proc. Symposium in Applied Mathematics*, J. T. Schwartz, ed., American Mathematical Society, Vol. 19, 1967.

[3] D. Gries, "Is Sometime Ever Better than Alway?", *Transactions on Programming Languages and Systems*, ACM, Vol. 1, No. 2, Oct. 1979.

——PETER WEGNER 編——

"Research Directions in Software Technology"

The MIT Press Series in Computer Science

The MIT Press, A 5変形判, xiii+869 pp., 1979

本書は MIT Press (Massachusetts 工科大学の出版局) が刊行しているコンピュータ・サイエンス・シリーズの2冊目である.

このシリーズでは今までに4冊が発刊されており，1冊目は，

1. J. E. Stoy "Denotational Semantics, The Scott-Strachey Approach to Programming Theory", 1977.

であり，標準的な denotational semantics の参考書として評価されている．3，4冊目はそれぞれ，

3. B. W. Arden, ed., "What Can be Automated? (COSERS) The Computer Science and Engineering Research Study", 1980.
4. S. A, Borkin, "Data Models: A Semantic Approach for Database Systems", 1980.

である．1冊目と4冊目がそれぞれ denotational semantics と data models についてのモノグラフであるのに対比して，本書と3冊目は共に研究のアセスメントを行った一種の調査レポートであり，大学の出版局が刊行するコンピュータ・サイエンスのシリーズの中で，本書のような調査レポートが発行されるのは異例である.

序文によれば，本書は，ソフトウェア技術の実践全般に対するソフトウェア研究の関連のアセスメントを行う目的で，Office of Naval Research, Air Force Office of Scientific Research, および Army Research Office がスポンサーとなって，Brown 大学の P. Wegner 教授を編者として，産学協同で「ソフトウェア技術全般の問題とソフトウェア研究の動向」をまとめた調査レポートである．その調査期間は1975年から1978年であった.

この意味で，本書は系統的に理論ないし技術自体を述べたものではなく，むしろ，理論や技術の評価をインフォーマルに述べている．したがって，ソフトウェアの研究・開発を管理する立場の人やソフトウェアの専門家でない人が，この分野についての1つの視座を得るのに役立つであろう.

さて，本書は2部構成で，

PART I : The Software Problem (問題の提起)

PART Ⅱ: Research Directions (研究動向)
からなるが，中心は PART Ⅱ に置かれており，PART Ⅱ はさらに，

PART Ⅱ,1: Software Methodology
PART Ⅱ,2: Computer Systems Methodology
PART Ⅱ,3: Application Methodology

に分けられている．各部分は，それぞれ数編の論文と論点をめぐる紙上のパネル討論から構成される．とくに，評価をめぐるパネル討論の部分が興味深い．逆に，各論文は，新鮮さに欠けるきらいがある．各部分を見ると，

PART Ⅰ: Software Problem では，ソフトウェア全般の問題提起を行う．総論的で，多分に“米国的発想”で問題点が述べられるが，管理の問題が主体となっており，ソフトウェア開発の労務管理を行う人に参考となろう．具体的事例として，Multics の経験が加えられている．PART Ⅰには，

1. B. W. Boehm, “Software Engineering: R & D Trends and Defense Needs”
2. H. D. Mills, “Software Development”
3. L. A. Belady and M. M. Lehman, “Characteristics of Large Systems”
4. F. J. Corbató aud C. T. Clingen, “A Managerial View of the Multics System Development”

の4編の論文がおさめられている．

PART Ⅱ,1: Software Methodology では，ソフトウェアを対象として，いわゆるプログラミングの技術が論じられている．PART Ⅱ,1 には，

1. C. L. McGowan and R. C. McHenry, “Software Management”
2. D. Gries, “Current Ideas in Programming Methodology”
3. B. H. Liskov and V. Berzins, “An Appraisal of Program Specifications”
4. R. L. London, “Program Verification”
5. J. B. Goodenough, “A Survey of Program Testing Issues”

の5編の論文がおさめられている．C. L. McGowan と R. C. McHenry の論文はソフトウェアの開発過程に関する問題を扱っており，他の4編とは色彩が異なる．

PART Ⅱ,2: Computer Systems Methodology では，ソフトウェア技術の支援系の問題が論じられており，

1. P. Wegner, “Programming Languages–Concepts and Research Directions”
2. P. J. Denning, J. C. Browne, and J. L. Peterson, “The Impact of Operating Systems Research on Software Technology”
3. J. B. Dennis, S. H. Fuller, W. B. Ackerman, R. J. Swan, and K. S. Weng, “Research Directions in Computer Architecture”
4. J. C. Browne, “Performance Analysis and Evaluation: The Connection to Reality”
5. R. E. Bryant and J. B. Dennis, “Concurrent Programming”
6. J. A. Stankovic and A. van Dam, “Research Directions in (Cooperative) Distributed Processing”

の6編の論文がおさめられている．

PART Ⅱ,3: Application Methodology では応用分野での研究動向を論じている．応用分野としては，数値計算，データベース，人工知能がとり上げられている．

1. J. R. Rice, “Software for Numerical Computation”
2. M. Hammer, “Research Directions in Data Base Management”
3. R. O. Duda, N. J. Nilsson and B. Raphael, “State of Technology in Artificial Intelligence”
4. R. C. Shank and W. Lehnert, “Review of Natural Language Processing”
5. M. Hammer and G. Ruth, “Automating the Software Development Process”

の5編の論文がおさめられている．

全体で900ページ近い大部なレポートとしては，おそらくは準備期間の問題もあってか，構成に多少難点が見られる．たとえば，PART Ⅰ と PART Ⅱの対象の関係，C. L. McGowan と R. C. McHenry の論文の位置等に，扱う対象と概念の不首尾が指摘できよう．しかし，全体で20編にのぼる著名な研究者による論文とパネル討論は，個別に見ると極めて優れたチュートリアルである．とくに，PART Ⅱ は優れた紙上のシンポジウムとなっている．

本書のタイトルどおりに，ソフトウェア技術の研究方向を占うという意味で，調査レポートとして本書を見ると多分不満が残るであろう．

カレンダ

◆第8回画像電子学会全国大会——東京，1980年6月5日～6日

37編の報告が発表され，260名の参加者があった．本年の特別講演は理化学研究所の出沢正徳氏による「モアレと計測」であり，この他「オフィス・オートメーション」などの2つのシンポジウムが開かれた．

◆SIGGRAPH '80——Seattle，1980年7月14日～18日

ACM SIGGRAPH (Special Interest Group on Computer Graphics) の主催で開かれた．今回はテレビジョン業界におけるコンピュータ・グラフィクスの適用など，4つのパネル・セッションが行われた．またテクニカル・プログラムの冒頭に「コンピュータ・グラフィクスのための有効な情報表現技術」と題して，グラフィック・デザイナをパネリストにして，現在のコンピュータ・グラフィックス技術を評価するとともに，ケース・スタディの結果を紹介した．取りあげられたテーマとして，高性能グラフィック・システム，曲面処理，CAD/CAM，アニメーション，地形情報データベース，大気環境モデル，電波天文学，教育への適用などがあった．

◆第4回生産技術に関する国際会議——東京，1980年8月18日～22日

本会議 (ICPE, International Conference on Production Engineering) は精機学会の主催による．基調テーマは生産技術の将来像とそれに対するアプローチを明らかにすることであった．

◆COMPCON '80 Fall ——Washington D.C., 1980年9月23日～25日

「分散処理」を主テーマに約40のセッションと100件を越える論文が発表された．席上，L. Rabinter と R. W. Schafer の両博士は，ディジタル音声処理およびディジタル・フィルタ設計における業績により，E. R. Piore 賞が授与された．基調講演は両博士によって行われた．パネル討論のテーマは，分散処理システムにおける人間的要因，UNIX の将来動向，テキスト処理言語の標準化，データ端末制御装置と回線終端装置とのインタフェース等である．この他，「科学的プログラマ対非科学的プログラマ」と題した討論があった．

Sperry Univac 社関係では，H. A. Freeman 博士が「ローカル・コンピュータ・ネットワーク」を入門的に解説したほか，M. L. Schneider, J. Larson, G.M. Schneider, H.A. Freeman, T.A. Welch らが，それぞれ「分散処理システムにおける人間的要因」，「スキマー併合のための再構成手順」，「効率評価」，「コミュニケーション・システム」，「多重コンピュータのアーキテクチャI」のセッション・チェアマンを務めた．また K. Thurber の「ネットワーク・アーキテクチャのステータス評価」および「ローカル・ネトッワークのハードウェアに対する調査」，J. A. Larson and T. B. Wilson の「分散データベースのためのデータ体系」，A. R. Hevner and G. M. Schneider の「分散データベース・ネットワークのための一貫した設計システム」J. P. McGovevr and D.Basn の「OSI（開放型システム間相互接続）のセッションとトランスポート」の5つの論文発表があった．

◆COLING '80——東京，1980年9月30日～10月4日

本会議（第8回計算言語学国際会議，8th International Conference on Computational Linguistics）は計算言語学国際委員会 (ICCL) の主催で開かれ，今回がアジアで初めての開催である．パネル討議のテーマとしては，計算言語学と言語理論，用語論と用図論，教育，機械翻訳が選ばれた．また，一般講演の主な話題は，辞書データベース，意味と知識表現，モンターギュ・システム，シソーラスの自動生成，かな・漢字処理，表意文字処理言語，データベース・アクセス用の自然言語，機械翻訳，ダイアログ，意味処理，コンコーダンスの作成，詩論へのコンピュータ適用などであった．

なお，Sperry Univac 社からは，N. K. Sondheimer が「不整入力に対する規則準拠アプローチ」と題して発表した．

◆FTCS-10——京都，1980年10月1日～3日

IEEE の Computer Society の フォルト・トレラント・コンピューティング技術委員会の主催するもので，今回アジアで初めて開かれた．17か国，188編の論文から73編が選ばれ発表された．主な話題は，商用システム (UNIVAC 1100/60 など) の信頼性技術，エラー訂正コード，主記憶のエラー訂正，設計の検証，OS との同期，ソフトウェア障害の回避とトレランス，LSI の テスト，確率的アプロー

チによるテスト，信頼性評価モデル，テストの網羅性評価，自動チェック，システム・レベルの診断，テスト・ケースの生成，分散型システム・アーキテクチャの信頼性，フォルト・トレラントな論理設計，専用システムにおける信頼性設計などである．

なお，Sperry Univac 社関係については，本誌96ページに掲載した．

◆電気4学会連合大会——東京，1980年10月2日～4日

特別講演は，根橋正人氏（電子計算機基本技術研究組合）による「超 LSI プロジェクトについて」と，柴田和雄氏（理化学研究所）による「光合成とバイオマス」であった．なお，パネル討論は「最近の超LSI技術」，「代替エネルギー開発の将来性」をテーマに行われた．また，主な話題は，大容量スタティック・メモリ，高速ゲート・アレー，VLSI 設計支援とテスト技術，光回路素子の現状，データ通信システムの故障診断，漢字自動認識，暗号における情報保護と認証・署名，音声合成システムの実用上の諸問題，話者認識，などであった．

◆第8回世界コンピュータ会議 (IFIP CONGRESS '80) ——東京，1980年10月6日～9日

世界44か国の約2,000人の専門家が参加し，91編の論文が発表され，17のパネル討論が行われた．情報処理基礎理論から日常生活におけるコンピュータ利用までの10部門に分かれて開かれた．また，パネル討論のテーマは，プログラムの仕様・検証・合成，並列処理コンピュータ・アーキテクチャ，CAD における人間と機械の関係，Ada 言語の設計目標と初期経験，超高水準言語と仕様言語，大規模データベースの理論・実際・諸問題，などであった．

また，主な招待講演のテーマは，ウィーン開発手法 (VDM) による関係データベースの仕様記述，非決定性プログラムの意味論，正規言語理論の発展，プログラムの正当性証明への様相論理の適用，ハードウェア・アーキテクチャの動向，記憶素子の動向，VAX の設計思想，ゲーデル解釈によるプログラム合成，関係データベースの現状，目標指向要求仕様定義，分散処理システムにおけるプロセス間コミュニケーション，テキスト・サービスおよびメッセージ・サービスの現状，データ通信とコンピュータ・ネットワークの現状，関数計算における任意精度のアルゴリズム，個人用代数演算機械，線画データの表現法，自動パターン認識，地域計画のためのシミュレーションにおける会話型最適化システム，体積モデル (Volumetric Model) による CAD システム，情報システムの分類と社会構造の変化，オフィス・オートメーションの研究と実用化とのギャップ，情報処理技術の社会的影響(合理化とモデル化)，プログラマに対するコンピュータ・サイエンス教育，幼児の心理発達に及ぼすパーソナル・コンピュータの影響，会話理論と一体感の限度，文書処理システム Hypertext，パーソナル・コンピュータ SMALLTALK のユーザ・インタフェースなどであった．

なお，当社の関係では，永田守男氏(慶大)，藤掛保隆（日本ユニバック(株)）ほかの論文「関数型再帰的プログラミング用の会話型支援システム」が発表された．

◆ACM '80 ——Nashville, 1980年10月27日～29日

「コンピュータ・エージを先見する (Previewing the Computer Age)」をテーマとし，A. Weinberg 氏が基調講演を行った．本年のチューリング賞は C. A. R. Hoare 教授 (Oxford 大学) に与えられた．技術プログラムでは，ACM の各 SIG (Special Interest Group) の主催により約45の論文発表・パネル討論，チュートリアル・セッションが開かれた．パネル討論のテーマは，自動情報検索システムの最近の話題，企業内ソフトウェア要員に対するコンピュータ・サイエンス教育，政治活動における情報管理，ホワイト・カラーの生産向上，コンピュータ性能評価，西洋文明のコンピュータ化による破壊，ドキュメンテーションなどであった．

一般講演の主なテーマは，分散型二重グループ・コンピュータ・ネットワーク(DDLCN)，プログラム・サイズに関するソフトウェア科学的分析，プログラミング言語における抽象化，プログラムおよびデータベースの構造的設計，プログラミング言語 (Ada の検証，データ・フロー・プログラムにおける命令参照パターンなど)，研究データ管理のための計算技術，小型機における芸術とインテリア・デザインにおけるコンピュータ利用，データベース・システムにおけるフォルト・トレランス・アーキテクチャ，ソフトウェアの社内共有のためのドキュメンテーションなどであった．このほか，Ada をめぐる最近の話題や ANS MUMPS の使用状況とその80年代の標準案などが発表されるとともに，PASCAL ユーザ・グループのミーティングや BIAIT (Business Information Analysis and Integration Technique) のシンポジウムが開かれた．

なお，Sperry Univac 社関係では，B. G. Claybrooks，H. Freeman がそれぞれ「モジュール：抽象データ型を定義・実現するためのカプセル化機構」，「データベース・コンピュータへのもう1つのアプローチ」と題して報告がなされた．

ABSTRACTS

——Univac 関係の論文・講演等の要約——

●ソフトウェアの設計・製作についての考察——ソフトウェアの生産においては，設計時のキーマンが製作時のリーダとなるのが理想的である．実際には配置転換によりキーマンがその仕事から抜けることが多い．本報告では要員の配置転換および設計詳細化の程度がソフトウェアの品質のに及ぼす影響を定量的に分析する．そして要員の配置転換に起因するソフトウェア品質の悪化を，設計の詳細化により軽減する可能性を示す．また，この設計の詳細化の程度を示す指数として，コーディング行数と設計仕様の行数との比を用いる．その他，本報告では並行処理における相互排除 (mutual exclusion) を記述する NS チャートのための新記号を提案する．(T. L. C. Chen, "Reflection on the Implementation of a Software Design", COMPSAC'79, Nov. 1979)

●言語ファイルとデータベースにおける入出力インタフェースの統合——現在，言語ファイルとデータベースにおいては，ふつう2種の入出力インタフェースが用意されている．この2つの入出力インタフェースの統合は，信頼性・保守性の向上，データベース関連の諸機能（データ辞書システム，アクセス制御・データ完全性など）の言語ファイルへの適用などの利点を持っている．本報告では，①データベースにファイル・アクセス機能を拡張，②ファイルにデータベース・アクセス機能を拡張，③ファイル管理システムとデータ管理システムを統合し単一のデータ管理システムを開発，の3ケースの各利点について論じる．さらに，言語ファイルの特質，言語ファイル定義・操作文を CODASYL のデータベース定義・操作コマンドに変換する方法・変換過程などについて述べる．(J. D. Lawrence, "Language and Data Base—Two I/O Systems or One?", COMPSAC '79, Nov. 1979)

●多重プログラミング下での各種のプロセス管理アルゴリズムを評価——ページングを採用した仮想記憶システムのスケジューリングおよび多重プログラミング制御では，ページ管理法，システム資源の利用ページのスラッシングなどを考慮する必要がある．本報告は仮想記憶環境におけるプロセス管理および，多重プログラミング・セットへのプロセスの投入管理のアルゴリズムについて論じる．(K. V. Sastry, "Process Management in a Paging Machine", COMPSAC '79, Nov. 1979)

●高移植性を考慮した高水準言語コンパイラの実現——多数のターゲット・システムに対するコードを生成する高移植性 COBOL クロスコンパイラを開発した．このクロスコンパイラは PLUS 言語で記述されており，現在すでに UNIVAC 1100 に移植されている．

コンパイラをデータ部と手続き部に分け，データ部については初期化の配慮や語長情報の利用，手続き部についてはポインタのインデックスへの変更，数値文字データの2進数への変換，OS とのインターフェース・ルーチンの用意などを行い，高移植性を実現している．そして，このコンパイラでコンパイルされたユーザ・プログラムの目的コードについても，インタプリティブな目的コードの採用により高移植性を達成している．この方式の利点として，①目的コードが非常にコンパクトであり，少ない記憶領域で実行できる，②新ターゲット・システムにユーザ・プログラムを移植する際はインタプリタのみを開発するだけでよい，などがある．(F. P. Mehrlich, "Portability-High Level Language Implementation", SIGPLAN NOTICES, Vol. 15, No. 1, Jan. 1980)

●UNIVAC 1100 用の LISP インタプリタを開発——この LISP の特徴は，インタプリタであること，使用の容易性およびコンピュータ資源の有効利用に力点を置いていることにある．本報告では，この LISP インタプリタに関する使用の容易性（括弧の個数のカウンティング，引用の容易性，各種のインタプリタ，文字セットとシフト・キーの制御など），構造化プログラミングとの関係，演算操作，論理的拡張 (CAR と CDR による合成，数，SETQ 目的関数)，OS へのシステム・インタフェース，エラー処理，Tektronix 4000 シリーズ・グラフィック端末を利用したアプリケーションのための組込みサブルーチンなどについて述べる．(R. M. Firestone, "An Experimental LISP System for the Sperry Univac 1100 Series", SIGPLAN NOTICES, Vol. 15, No. 1, Jan. 1980)

●ネットワーク定義および分析のための言語 NDL を開発——DCA に基づくコミュニケーション・ネ

ットワークのトポロジカルな構成を定義する高水準手続言語を開発した．この言語 (Network Definition Language)は次の３つの部分からなる．宣言文ではネットワークのコンポーネントを，関係文ではコンポーネント間の通信回線を，手続き文ではネットワークのトポロジーが変更される場合の条件とその場合のアクションなどが定義される．この言語の目的はシステム生成を容易にするためにある．NDL で記述されたネットワーク定義は，NDP (Network Difinition Processor) で処理され，機器構成ファイルとして出力される．そしてこのファイルは，ネットワーク・マネジメント・サービスにより利用される．(M. A. Mayor, "A Language for Network Analysis and Definition", SIGPLAN NOTICES, Vol. 15, No. 1, Jan. 1980)

●構造化プログラミングへの新しいアプローチとして，ゲーデル数によるプログラミングの採用——ゲーデル数によるプログラミングとは，ゲーデル数により計算対象の入出力仕様を規定するもので，実際の計算手順そのものを規定するものではない．また，ゲーデル数を特定のコンピュータの機械語に変換するためには，最適化機能を持つコンパイラが利用される．この方式には，①入出力仕様設計と内部プロセス設計の分離ができる，②入力が数字だけなので，構文解析が不要である，③同様の理由により端末機器の価格が安くなる，④プログラムのバージョン管理が容易になる，などの利点があげられる．(N. H. Cohen, et al., "Gödel Numbers: A New Approach to Structured Programming", SIGPLAN NOTICES, Vol. 15, No. 4, Apr. 1980)

●テスト・データ選択における新方法を提案——テスト・データの選択基準の信頼性および妥当性を証明することは，プログラムの正当性の証明と等価で，非常にむずかしく，実用性に難があった．そこで，エラー開示テスト基準 (Revealing Test Criterion) とエラー開示サブドメイン (Revealing Subdomain) の概念を導入する．たとえば，プログラムの入力ドメインのサブセットであるサブドメインが開示的 (Revealing) であるとは，「サブドメイン中に１つでもプログラムをエラーに導くデータがある場合には，そのサブドメインの部分集合をテスト・データに用いると常にエラーが発生し，一方，プログラムが正しく処理されるデータが１つでもある場合には，サブドメインの部分集合をテスト・データとすると，常に正しい結果が得られること」と定義する．プログラムの入力ドメインを，開示的サブドメインに分割し，それから１つずつデータを選択してテストすれば完全なテストが可能となる．しかし，サブドメインがすべてのエラーについて開示的であることを証明するのはむずかしいため，特定の１つのエラーに対し，開示的であるようなサブドメインを用い，入力ドメインを構成し，各サブドメインから選択されたデータを用いテストを行うのが，現実的な方法である．(T. J. Ostrand, et al., "Theories of Program Testing and the Application of Revealing Subdomains", IEEE Trans. on Software Engineering, Vol. SE-6, No. 3, May 1980)

●主記憶の割付け制御によるスケジューリング・アルゴリズム——異なるクラスの複数のユーザが同時に使用する仮想記憶コンピュータ・システムでは，クラスごとに決められたサービス目標を満たす必要がある．報告されたアルゴリズムは，まず，主記憶の予想使用量のみに基づき主記憶のバランス条件を満足するプロセスを活性化 (activate) する．次に，資源の利用についてクラス間でアンバランスが発生するときは，資源使用のアンバランスの程度と種類に基づき，あるクラスとそのクラスに属するプロセスを選択し不活性化 (deactivate) する．このアルゴリズムは，本質的に自動修正機能を持つため，目標へのプロセスを動的に調整することができる．(K. V. Sastry, "Balancing Processor Shares of Scheduling Class through Controlled Allocation of Memory", NCC '80, May 1980)

●大学のコンピュータ関連学科におけるコンピュータ・ハードウェアのカリキュラム——ハードウェアについての教科には，コンピュータ・アーキテクチャとコンピュータ・オーガニゼーションがあり，前者はプログラマから見たハードウェアの機能的構造に関するもので，後者はあるアーキテクチャの１つの実現について学習するものである．コンピュータ・オーガニゼーションの基本教材としては，仮想的機械を用いるよりも，現実の商用コンピュータを単純化したものをケース・スタディする方法が実際的である．また，今後は複数のマイクロプロセッサを用いるシステムが普及するため，回線交換・パケット交換・バス構造などの接続技術を教科として取り入れる必要がある．(K. J. Thurber, "Teaching Computer Structures", IEEE Computer, June 1980)

●コンピュータ援用による寸法公差の自動決定アルゴリズム——本報告は，最終的公差を許容範囲内におさめ，加工コストを最小にするよう，機械加工の各段階での寸法公差を決定する問題である．この問題を解く困難さは，寸法公差を決定するための目的

関数や制約式となる“加工コストと公差との関係式”が非線形な点にある．アルゴリズムとしては，加工コストと公差との関係式の線形化を行うか否か，統計的品質管理の実施を前提とするか否かなどの相違により分けられる．大規模な問題では線形計画法が，小規模な問題では Lagrange乗数法と Newton-Raphson 法のなどが利用できる．(A. M. Patel, “Computer-Aided Assignment of Manufacturing Tolerances”, 17th Design Automation Conference, June 1980)

●LSI設計の自動化問題における計算の複雑性について分析——設計実施（合成，置換え可能なモジュールからなる標準ライブラリの構築，階層分割，モジュールの選択），配置設計，配線設計，障害検出などにおける最適設計の問題の多くは，Steiner トリー，巡回セールスマン，クロマチック・ナンバー，ナップサックなどの “NP 困難な問題” に帰着される．そこで，ヒューリスティックスが重要となり，これらの問題を解くために確率的分析や実験，高度な並列処理アルゴリズムの研究などが行われている．本論文では，設計実施・配置・配線・障害検出における諸問題の定式化と複雑性の解析を行う．(A. Bhatt, “The Complexity of Design Automation Problems, 17th Design Automation Conference, June 1980)

●音声応答システム等における任意複合数詞音声の合成方式の提案とその評価実験——現在，音声出力で問題になるのは音声の合成方式である．任意の数字を組み合わせて複合数詞として位取りを行い，アクセントを付加して自然な音声を出力するには工夫を必要とする．本講演者は日本の共通語（いわゆる標準語）の数詞結合則の中からコンピュータによる音声合成に適したアルゴリズムを抽出し，それを用い任意の複合数詞にアクセントを付与して発声させる実験を行い，その実用性を確認した．本報告は，この実験で用いた複合数詞音声の合成アルゴリズムと実験結果について述べた．（若鳥陸夫，計算言語学研究会，1980年6月講演）

●漢字列長単位用語の抽出法と方法——分かち書きに使われる用語の収集・研究は，その分割の大きさにより短単位と長単位に大別される．短単位用語は辞書の編集，語の意味分析，語基の研究のために有用であり，長単位用語は機械翻訳および専門用語（熟語）を取り扱うときに便利である．本報告では機械支援による翻訳（完全な機械翻訳も含む）においてよい訳語を与え，用語の多義性を減らすことに役立つ長単位用語の収集方法を考察する．そしてその具体例として同氏が試みた用語収集の自動処理を紹介する．この方法は JICST のファイルを利用し網羅的に漢字列を収集するものである．（田中康仁，計算言語学研究会，1980年6月講演）

●自然言語理解システムにおける不整入力の処理方法——しばしば，綴りや文法上のエラーを含むもの，全く意味をなさないものなどが入力される．このため自然言語理解システムでは，システム側がユーザの意図を推量することや，不明な点を明らかにするようにユーザに要求することが，必要となる．そして，最悪の場合でも入力の不整を正確に識別することが必要となる．ここで述べる方法は，規則準拠 (rule-based) アプローチともいうべきもので，入力が正しい (well-formed) と仮定して通常の処理規則により処理し，入力の不整により処理できない場合には，エラーのタイプごとにあらかじめ定められたメタ・ルール (meta-rule) により通常の処理規則を修正し，できるだけ完全に解釈するように処理を行う．この方法は，現在，世の中に存在する各種の不整入力処理方法を吸収できるばかりでなく，システム開発や処理効率の点で優れている．この方式を用いた実験システムが Sperry Univac社および Delaware 大学で構築中であり，Sperry Univac 社では，データベース・システムへの自然言語インタフェース（英語でフロント・エンドと呼ばれている），Delaware 大学では英語入力による質問応答システムが，開発途上にある．(N. K. Sondheimer, “A Rule-based Approach to Ill-formed Input”, COLING '80, Sep. 1980)

●OR におけるコンピュータの利用技術——コンピュータ白書の適用業務の調査データからもうかがえるように，近年，コンピュータは事後処理業務から予測などを含む事前処理業務へと，適用範囲を拡大している．このため，事前処理業務に有効な科学的技法を提供する OR の重要性が増大している．これまで，OR の研究成果は個々の問題ごとのアプリケーション・ソフトウェアという形態で，情報システムに取り込まれてきた．しかし，事前処理業務で対象とされる問題は，本来，非定型的・非定常的性格を持ち，既知の簡単な問題に還元しにくいため，人間の持つ創造性を生かしたマン・マシン・システムが必要となってきている．そのシステムでは，必要な時に必要な情報が検索でき，必要に応じて各種の加工技術（OR 手法）が使用でき，その結果を見やすい形で提供することが求められよう．（小林弘和，日本オペレーションズ・リサーチ学会，1980年秋季研究発表会，特別講演，1980年10月）

述にとどまらず，ソフトウェアのライフ・サイクル全体の段階を支援できるように，Michigan 大学の D. Teichrow 教授による要求仕様言語 PSL/PSA を拡張したデータベースを持つ言語系で，現在オペレーティング・システム等の開発を支援している．以上は，ソフトウェアの研究・開発・保守のサイクルを支援する実践技術を述べている．

G. A. Champine の**データベース・システムの最近の傾向**は，データベースのソフトウェア，ハードウェアについての技術のアセスメントを行う立場から，平易にデータベースに関する全般的な技術動向の紹介を試みたものである．とくに，マス・ストレージとデータベース・コンピュータの開発に関連した部分を含む．T. A. Welch の**記憶域階層構成の解析**は，Champine の論説と表裏の関係にあり，古典的な命題でもあるコンピュータの記憶装置の費用を勘案した最適設計の基礎を論ずる．主としてバックエンドの情報システムでのキャッシュ・ディスク等のマス・ストレージを最適なバランスで構成するための基礎研究の１つである．この意味で，古典的命題であるが，費用とスピードを考慮した適正な記憶装置を構成することは，とくに現在の重要なテーマである．G. S. Tjaden と M. Cohn との**マイクロプロセッサ技術によるプロセッサ設計の考察**は UNIVAC 1100/60 (Vanguard) システムで採用したマイクロプロセッサによる主プロセッサの設計を述べている．彼らは 1975 年を中心に大きい主プロセッサをマイクロプロセッサを使って設計する，主として設計上のファクタの最適性の研究に従事した．中心となる中央プロセッサをマイクロプロセッサを使って組むのは商用の大型のコンピュータでは初めての試みであり，今後のシステム開発の１つの方向を示すものであろう．藤野・渡部の論文と長島の報告は応用ソフトウェア分野のアルゴリズムの比較評価と改良に関するものである．藤野・渡部の**新しい差分法の提案と数値実験**は，古典的な差分法をとりあげて，正則条件を満たす直交曲線座標系上の差分法と FEM の比較評価を行うために，流体の問題を事例として実践した結果を中間的に発表する．長島の**回路解析プログラム CIRCUIT の特徴と数値計算技法**は，長島が過去，回路設計用ソフトウェア CIRCUIT に実装してきたアルゴリズムの改良をまとめて論じている．長島は R. Brayton 氏らのアルゴリズムと長島が改良した疎行列操作による改良型 Gustavson アルゴリズムを採用して，従来の回路解析用アルゴリズムの効率化を実現した．（編集子）

EDITORS' NOTE

技　　報

UNIVAC TECHNOLOGY REVIEW

No. 0

発　行　日　昭和 56 年 2 月 1 日

発行人兼編集人　富 田 和 夫

発　行　所　日本ユニバック株式会社
東京都港区赤坂 2-17-51 〒107
TEL (03) 585-4111 (代表)

印　刷　所　三美印刷株式会社