講　　座

プログラマのための電子計算機雑話（I）*

石　井　康　雄**

はじめに

電子計算機は急速な発達をしている．そこで，これに対する考え方も広く深いものになっている．計算機について，どのように考えるべきかという問題は，重要問題の一つであることは，間違いがない．

そこで計算機に関連する基礎的な事実と，それに対する考え方を中心として，主としてプログラマのために，計算機の解説を試みることにした．

電子計算機について考えるべき問題は，あまりにも多いので，思いついたことから，逐次検討するという形で，問題をとりあげてゆくつもりである．

計算機についての基本的な用語，たとえば，入出力装置，ファイル，コンパイラ，あるいはアドレスなどという言葉は，一応知られているものと仮定して，話しをすすめることにする．

1.　電子計算機の規模

1.1　グロシュ（Grosh）の法則

電子計算機の規模に各種あって，慣用的に大型機，中型機，小型機などと呼ばれている．これらの区分を厳密に定義することは，統計資料の比較などの場合を除くと，大して重要ではない．しかし，電子計算機をこのように区分してながめてみることには，意味がある．

計算機のコストとその能力について，よく知られている経験則があってグロシュの法則と呼ばれている．これは，計算機のコストは，能力の平方根に比例するというもので，安い計算機を作ろうとしても，なかなかコストは下がらぬものだというようにも解せられるが，通常はコストの二乗に比例して性能が増大すること，すなわち，大型機になると，小型機を何台か合わせたものよりも，格段に高性能になると解している．

計算機のハードウェアが，不安定さを持っていたころには，小型機2台と大型機1台が，もし，同じコストならどちらがよいか？　というテーマの検討が，行なわれたこともあった．現時点では，この問題は，もう片づいてしまったといえるであろう．

なぜ，大型化すると急速にパフォーマンス対コストが向上するのかというと，これには，いくつかの理由が考えられる．

（1）　装置の一部分の性能を強化することによって，それ以外の部分の効率を高めうること．

たとえば，充分高速の本体に入出力装置が，1組しかついていないとする．ここで，入出力装置をもう1組追加すれば，他をふやさなくても，全体の処理能力は，ほぼ2倍になるであろう．処理能力のネックになっている部分を強化することが，要点であるのはいうまでもない．

（2）　記憶容量を大きくすることによって，補助記憶とのやりとりをへらすこと．

コアの中だけでできる処理と，補助記憶を使わなければできない処理とでは，能率に大きい差がある．そこで，コアを大きくするだけで，能率が 10 倍以上高くなることがある．

（3）　ランダムアクセスファイルを増設することによって，低速入出力装置の動作時間を吸収したり，入出力装置を増設することによって，人手操作の時間を吸収できる．

（4）　以上のことを行なう場合，大型化したシステムにおいては，重要な点の改良のためのコスト上昇が，全体としてはあまりひびかない．

1.2　大型化の限界

大型機と小型機の相違は，各装置の機能・性能・操作など，たとえば，演算速度と演算精度，記憶容量と読み書き速度などの諸点で，大型機がすぐれている点にある．また，大型機ではそれらの諸点が，さらに改

* An introduction to basic problems of computer systems, by Yasuo Ishii (Nippon software CO., Ltd.)

** 日本ソフトウェア(株)．

善される可能性の幅も大きい．そして，すでに触れたように，その性能をより高めるための投資効果が大きい．

この限りにおいて，大型化は明らかに有力であるが，大型化するほど不利になる要因もある．

（1） ハードウェア的技術が，あまりにも複雑化する場合．この場合は，もし，障害がおこったとすると，その回復のために，加速度的に手間と時間がかかってくる．

（2） システムが過大になると，活用するための手続きが複雑になる．たとえば，システムの資源配分や，マルチプログラミングの制御に工夫を要する．

（3） 使用法が複雑になると，システムを制御している管理プログラムの負担が増大し，管理プログラムの運用コスト（オーバヘッド）がふえるとともに，管理プログラムの開発に，コストと人手と時間がかかるようになる．

（4） 情報の伝達には，物理的な速度限界がある．そのため計算機の性能を高めようとすると，むしろ，物理的には小型化しなければならない．これには部品と素子の面で，新しい進歩が必要である．

以上のような問題があるので，大型機には限界がある．しかし，新しい部品・素子の開発と，それに伴う新しいシステム構成によって，従来，とても考えられなかったことも，実現されるようになり，大型機の限界は，常に更新されてきた．

最高の計算機のコストは，これまで常に一定であった（ほぼレンタル換算 15 万ドル）といわれていることと，大型計算機の大きさは，常に小さくなろうとしていることとは，興味ある事実である．

2. 互換性

2.1 互換性の意義

電子計算機の設置台数が増加している．そこで，よその計算機のデータを利用したり，プログラムを共用することによって，むだな費用を防ぐことが期待される．

実際問題としては，これまで使っていた計算機を，新型機に置き換えようとするときに，それまで蓄積したデータやプログラムが，全然使えなくなることが大問題である．データやプログラムを，相互に利用しうることを，互換性があるといい，一方は他方を利用しうるが，逆は不可というときには，大きい方に対する上方互換性（Upward Compatibility）があるという．

互換性の重要性は，標準化の重要性といってもよい．互換性を保つためには，情報の媒体の物理的仕様，その媒体に記録する方式，および記録についての物理的仕様，媒体の記録の論理的な意味などを，正確に規定する必要がある．これらの規定について，国際的な活動をしている ISO (International Organization for Sandardization) の意義は大きい．

2.2 データの互換性

データは，カード，紙テープ，磁気テープ，磁気ディスクパックなどのデータ媒体に記録される．媒体そのものの物理的仕様，その媒体に記録するときの精度や記録位置，記録の方式，記録をデータと対応させるときの符号の意味，記録を読みとるときの精度・方式などが規格どうりになっていること，したがって，一方が記録したデータを他方で読みとれるということだけでは，充分な互換性を持っているとはいえない．以上の点は，互換性の最低限度であって，最近の要求はよりきびしい．すなわち，データを読みとってから処理するまでの手間が，ほとんどないかどうかを問題にするようになってきた．データが単純な構造——文字・数字・記号などのならびで構成されている——だけではなくなってきて，データの一部に，データの構造に関する情報も含めることが，行なわれるようになったため，ある計算機で使われている構造のデータを，他方の計算機ではスムーズに処理できないという現象をおこすようになってきた．そこで“とに角，処理しようと思えば可能”という状態の互換性を“ハードウェア上の互換性”と呼び，読みとったデータを，もとの計算機の処理手順と等価の処理手順で処理できる状態の互換性を“ソフトウェア上の互換性”まで考慮されているというように表現する．ハードウェア上の互換性しか持たない場合は，情報変換のための（余分な）プログラムを用意しなければならない．

2.3 プログラムの互換性

プログラムの互換性，すなわち，一方の計算機のためのプログラムが，他方の計算機にも，そのまま使えることは，新旧計算機の交替時において，プログラムの蓄積が大きければ大きいほど重要である．

プログラムの互換性には，二つのレベルがある．一つはコンパイラ言語，すなわち，プログラムの記述言語のレベルでの互換性であって，たとえば，FORTRAN, ALGOL, COBOL, あるいは PL/I などの言語レベルでの互換性である．通常，コンパイラ言語のレベルでは，互換性があると強調されているが，実は，厳密に

は問題がある．しかし，ほとんどの場合に，ちょっとした手直しで互換性を回復できるので，大きな問題にはならないことが普通である．しかも，計算機が新しくなり，コンパイラが新しくなると，もとのプログラムに手を加えなくても，コンパイル（機械語相当のレベルに変換すること）を改めて実行するだけで，全体のプログラムが高能率になる．

これに対して，アセンブラ言語のレベル，ないしはコンパイラ言語のオブジェクトのレベルでのプログラムの場合は，本来が計算機の構造に依存することの強いレベルの言語であるので，計算機が新しいものになると，多くの場合に，古いプログラムは生かすことがむずかしい．いいかえれば，プログラムの書直しが必要である．もし，新しい計算機と古い計算機との間で，同じオブジェクトプログラムを利用できるならば，機種変更の抵抗は著しく小さくなる．

第2世代の計算機では，オブジェクトプログラムのレベルでの互換性は，相互にないのが普通であったし，また第3世代の計算機との互換性もない．しかも，第2世代のころは，コンパイラの普及率が低かったために，コンパイラ言語の利点も利用できないことが多い．

第3世代になって，この二つの点は著しく改善されたので，互換性の問題は大幅に前進したと考えられる．

なお，プログラムの互換性のある二つの機種の間でも，古いプログラムの書換えは，一般には必要である．なぜなら，小型の機種のためのプログラムは，大型機の性能を充分に活用できないことが多いからである．互換性がないと大変だということは，機種切換の時点で，一挙にプログラムの作り変えを終えなければならないという点であって，互換性が保証されていれば，いずれは，全部のプログラムを作り変えなければならないとしても，ゆっくりと時期をみて，作り直す余裕を持ちうる．この余裕が実用上有効である．

3. ソフトウェアとの相補性

3.1 概　要

小型電子計算機では，浮動小数点のためのハードウェア回路を持たないものが多い．浮動小数点計算をしたいときは，サブルーチンを使う．これが大型計算機になると，当然のように浮動小数点は，ハードウェアで処理される．このように，一つの機能をハードウェアで具現することも，ソフトウェアで満足させることもできることがある．これをハードウェアとソフトウェアの相補性と呼ぶことにする．

ハードウェアの基本システムにある付加装置を追加して，機能強化をはかることがある．付加装置なしで，その機能を利用したいときは，ソフトウェアによって実現する（相補性の活用）．

ある機能をハードウェアで与えるときは，そのコストは計算機使用料の増加となって現われる．それに対して，ソフトウェアで与えるときは，そのコストは，処理時間の増加と，コア記憶装置の占有という形で現われる．コア占有という問題は，浮動小数点演算の機能などでは，ほとんど無視してもよいが，管理プログラムなどでは，かなり大きいものとなる．

3.2 ハード，ソフトのコスト比較

簡単のため，あるシステムの単位時間当りのコストを S とし，ある処理を実行するための時間を T とする．また，このシステムにある機能をハードウェアで追加するときの，単位時間当りのコスト上昇分を ΔS とし，また，その機能をソフトウェアで代行する場合に生ずる処理時間の増大を ΔT とする．

コア占有のコストは無視する．そうすると，その機能をハードウェアで実現したシステムのコスト（C_H）は

$$C_H=(S+\Delta S)*T \tag{3.1}$$

また，その機能をソフトウェアで代行したシステムの処理コスト（C_S）は

$$C_S=S*(T+\Delta T) \tag{3.2}$$

$$\therefore\ C_H-C_S=S*T*\left(\frac{\Delta S}{S}-\frac{\Delta T}{T}\right) \tag{3.3}$$

(3.3)式を考えてみる．もし，S が充分大きいとき，すなわち，大型計算機では一般に T は小さいと考えられるので

$$C_H-C_S<0 \quad (S \text{ が大})$$

これに対し，S が小さい値のとき，すなわち，小型計算機では一般に T が大きいと考えられるので

$$C_H-C_S>0 \quad (S \text{ が小})$$

となる．すなわち，大型機では一つの機能を，なるべくハードウェア化する方が有利であり，小型機では，その逆となることを示す．

3.3 ソフトウェア化の利点と欠点

ある機能を考えたときに，その仕様に未確定部分が避けられないときは，ハードウェア化が困難である．その場合は，ソフトウェアで処理しておけば，あとで仕様の変化があっても，影響が小さい．ハードウェアの改造は，工場を動員しないとできないけれども，ソ

フトウェアではプログラマが，自分で手直しすることができる．実は，このことがソフトウェアの大きな“わな”になるのであるが，うまく使えば，一つの特徴であることも確かである．

ソフトウェア化すると，ハードウェアを時間的なものに置き換えたことになるので，見掛け上安上がりにすることができ，小型になる．

ソフトウェア化の第3の特徴は，その機能の分析が容易であるという点である．ハードウェア化された機能は，ハードウェア側の技術者だけに，その分析が可能である．

ソフトウェアは，このような利点を持っているが，その本質は，命令を一つずつ実行することであるから，簡単なことでも，命令の読出しと解読という手間を伴うために，かなりの時間がかかる．しかも，命令は一つずつ行なわれるから，ハードウェアならば並列的に処理できることも，直列的に処理することになる．その他，要するに時間の要素で，かなりの不利を伴う．ハードウェアのコストが下がると，ハードウェア化が進むのは当然である．

4. スピードアップ技術

4.1 概 要

ハードウェアのシステムを考えるとき，その進歩は，要するに，スピードアップであったといっても，大きな誤りとはいえないであろう．

スピードアップの方法は，大体4通りある．すなわち

（1） 高性能の部品・機械を作ること
（2） 並行処理
（3） 先回り制御
（4） 処理方式の再検討

このうちで，素子の高性能化という方法は，システムの性能を高めるという目的からいえば最も望ましいものであり，当然のことながら，それは大きな実績を持っている．継電器時代から，最近の集積回路に至る発達を考えてみても，ハードウェアのシステムは，素子を無視して成立するものではない．スピードアップの技術として，先に述べた“並行処理”，“先回り制御”，“処理方式の再検討”というものも，ある程度をこすと複雑すぎて，制御できない状態になってしまうであろう．ハードウェアの内容が複雑すぎると，それを生かしていくハードウェアのシステムが設計困難となり，ハードウェアが処理能力の可能性を高めても，結局は速度の壁にあたってしまうからである．

そこで，これらスピードアップの技術は，充分にバランスのとれたものでなければならない．

4.2 並列処理

一般に，直列処理を並列処理することは，スピードアップの基本であり，逆にすれば，コストダウンの原則となる．

データの読出し，書込み，演算について，初期の計算機は1ビットずつ処理すること（直列処理）が普通であったが，やがて1文字（4～8 ビット）単位で並列に，しかし，全体としては文字単位，あるいは1数字の単位で，直列に処理する（直並列処理）ようになり，さらに 10 進 10 けた，あるいは2進 36 ビットというような語の単位で，並列処理することが大型機の普通の状態になった．同時に，数語まとめて読み書きすることも行なわれる．

加算器を複数個おいて，計算を高速化することも行なわれている．加算器に付随したレジスタを多数おくと，レジスタをクリヤしたり，演算のために被演算数をレジスタにセットしたりする手間を省くことができる．演算の並行処理の著しい例は，インデクスレジスタである．アドレス修飾用の加算（インデクシング）を先取りして実行すること，すなわち，一つ前の命令の演算を実行しているときに，インデクシングも並行して実行していることが行なわれる．インデクシングを普通の加算の命令で行なうためには，数個の命令が必要であるから，インデクシングをハードウェアで並行処理することによって，数倍はスピードアップしたことになる．

並列的に処理を行なう他の例は，チャネルの機能である．チャネルは入出力装置と記憶装置の相互間の，情報転送の制御を行なうもので，計算機本体から，入出力制御開始命令を受けとると，計算機本体の動きとは独立に，入出力制御を始める．処理内容の指示は，チャネルが記憶装置から読み出して解読し，その結果を入出力装置の制御信号とし，あるいは逆に，入出力装置の状態を識別して適当な制御を行ない，動作の完了やエラー発生などを本体に連絡する．入出力装置と記憶装置相互の情報の転送は，このようにして計算機本体のうごきと独立に処理される．現在は小さな計算機でも，高速の入出力のためには，チャネル機構を持つようになっている．もし，チャネルなしで，すべての情報転送を本体が処理するものとすれば，全体の能率は著しく低下することであろう．

すべて並行処理を行なうためには，処理すべき情報と処理結果とを，記憶すべき場所が必要である．入出力装置のための記憶場所をバッファと呼ぶ．たとえば，計算を本体で行ないながら，印刷を並行して行なうためには，印刷装置にバッファをおくのが普通である．チャネルの機能を生かすためには，コア記憶装置の一部が，バッファとして用いられる．なお，チャネルを一般化し，中央処理装置と同等の機能を持たせようとする考え方もある．

4.3 先回り制御

命令を実行している間に，次の命令の読出しと解読をすませておくということは，命令先取りの基本である．最近は，数個以上の命令を記憶装置から読みとって，できるだけの解読をすませてしまうということも，実用化されている．その場合，もし，分岐命令があれば，分岐する場合としない場合の両方を解読しておくことが行なわれるが，そこまでいくと，やや複雑の度が強すぎるとも考えられるが，要はバランスの問題である．

データの先取りも行なわれる．これは高速の記憶場所に，あらかじめ低速記憶場所からデータを読み出しておくことであって，高速レジスタとコア記憶，コア記憶とドラム，あるいはディスクという組合せで行なわれることが多い．最近は，レジスタに相当する記憶場所の容量を充分大きくして，一層の効果をあげようという考え方も行なわれている．

4.4 処理方式の再検討

スピードアップで重要なことは，むだな待合せを省くことと，どうしても避けられないときは，その頻度を減らす努力をすることである．

たとえば，ドラムやディスクなどのデータを読み書きするときに，データを1件ずつ読み書きすると，1件ごとに大きなアクセスタイムを必要とする．このようなときに，プレソートと呼ばれる手法がある．たとえば，数件のデータを読み出すことになったときに，読出しの要求のあるたびに，ドラムにアクセスしているとする．平均アクセスタイムを a，1件あたりの読書時間を b，要求件数を n とし，要求は比較的集中的に行なわれたとすると，処理時間は，ほぼ

$$n*(a+b) \qquad (4.1)$$

だけかかるであろう．これに対し，要求を n 件到着するまで待ってから，読み出すべきアドレスの順に要求を並べかえると，最良の場合には，ドラムを1周する間に（すなわち，$2a$ の時間で）すべての処理が終了する．

最悪の場合でも (4.1) 式と変わらない．したがって，一般にはかなり高速化されるであろう．

また，磁気テープの読書きは，段落ごとに一時停止をするので，読書きは一度になるべく多量の情報を扱うのがよい．テープの停止時間を 5 ms とし，読書きの速度を平均して $a\times10^3$ 字／秒とすると，N 字を読み書きするときの実効速度は

$$a\times10^3/(1+5a/N) \text{ 字／秒} \qquad (4.2)$$

となる．仮に $N=1,000$ 字，$a=50$ とすると，実効速度はテープが走っているときの読書速度の 80% になる．これと同様に，ドラムのアクセスタイムを仮に 20 ms とすると，読書速度を $C\times10^3$ 字／秒としたときに，実効的には

$$C\times10^3/(1+20C/N) \text{ 字／秒} \qquad (4.3)$$

となる．もし，$C=1,000$（すなわち 1 メガ字／秒）とし，$N-2,000$ とすると，実効速度は 9% に落ちてしまう．

バッファを A, B 2組設けておき，バッファAにデータを読み書きする間に，他方のバッファBのデータを処理し，データ処理が終わると，バッファAのデータを処理するとともに，バッファBの内容を書き出し，かつ，書出しが終了したら，新しいデータを読み込むものとする．この場合，もし，読み書きの時間とデータ処理の時間を，それぞれ R, W, P とすると

$$R+W=P$$

ならば，完全にむだなくデータの処理が行なわれたことになる．もし

$$R+W=nP$$

であれば，バッファを $2n$ 組設けておき，読書きのチャネルを n 本用意できれば，完全にむだなくデータの処理が行なわれたことになる．この簡単な例から，すべて，スピードアップの技術は，現実の数字を考えてみないと，最適の解答が出せないこと，いいかえると，一般的な最適解を求めることがきわめてむずかしいことが予想できる．　（つづく）