オペレーティング・システム 41－2
（１９８８．１１．１１）

並列コンピュータの
プログラム分割に関する基礎考察

伊藤　勤　　曽和将容

名古屋工業大学電気情報工学科

並列処理を効率的に行うために，”プログラムを，どのように処理の単位に分割し，プロセッサに割り当てるか．”という静的スケジューリングが非常に重要である．
本論文では，プロセッサ間の通信を考慮した分割法を考察し，この結果を具体的なプログラムグラフに適用して，分割法の評価を行ったので報告する．

A STUDY OF PARTITIONING PROGRAMS FOR PARALLEL COMPUTING

Tsutomu ITO　and　Masahiro SOWA

Department of Electrical Engineering and Computer Science
Nagoya Institute of Technology, Gokiso,Nagoya 466,Japan.

In the parallel processing, it is very important how to partition a program into the units of processing and allocate them on processors.
This paper analyzes the execution time of the programs which are partitioned by a method in which communication between processors is considered.

1. まえがき

一般に，プログラムの中には，同時に実行可能な命令が多く存在する．並列処理コンピュータは，それらの命令を，複数のプロセッサに割り当てて実行させることによって高速化を図る．これを実現するためには，プログラムをどのように処理の単位に分割するか，また，その処理の単位をどのプロセッサに割り当てて，いつ実行させるかが問題となる．このようなスケジューリングの問題については，これまでにたくさんの研究が行われてきている．スケジューリングの方法には，動的スケジューリングと静的スケジューリングがある．動的スケジューリングは，実行時にスケジューリングを行うので，その分だけ処理時間が長くなる．静的スケジューリングは，実行前にスケジューリングを行うので，動的スケジューリングに比べて処理時間は短い．しかし，あらゆる問題に対して，静的な最適スケジューリングをすることは，プログラムに非決定的な部分が存在することなどにより，困難な問題となっている．このため，静的スケジューリングの研究の多くは，特定の問題について進める傾向が強かった．

最近，非循環有向なプログラムグラフについて，処理の単位の大きさが最適になるように決定し，その処理の単位をスケジューリングすることに関する研究が行われている[4]．

われわれは，スケジューリングに関する上のような現状をふまえて，あらゆる問題に対して，より最適に近い静的なスケジューリングを行うために，プログラムをどのような処理の単位に分割するべきか，について研究を進めてきている[1),2)]．本論文では，プロセッサ間の通信を考慮しながら，いくつかの分割法について考察し，この考察の結果に従って，実際のプログラムグラフをプログラムグレインと呼ばれる処理の単位に分割した．そして，各々のプログラムグレインを１台のプロセッサに割り当てて実行させたときの処理時間を算出し，各々の分割法を使ったときの処理時間を比較することによって，どのような分割法がより最適であるかについて考察を行った．

2. 本研究における仮定

2.1 プログラムグラフ

図2.1に，並列処理コンピュータで実行されるプログラムグラフの例を示す．ここで，矢印は理論的には制御の流れを表している（物理的にはハードウェアによる通信を表している）．また，長方形の中に書かれているのは命令である．各命令は，固有の実行時間を持ち，実行に必要な通信が行われたときに実行が可能になる．

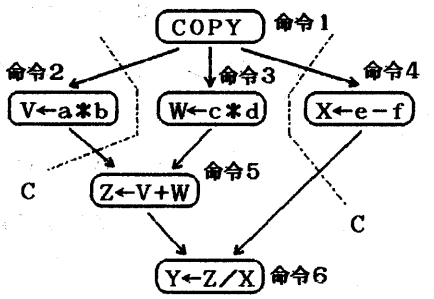

図2.1 プログラムグラフの例

2.2 プログラムグレインに関する仮定

図2.1を，破線Ｃで分割した例が図2.2であり，プログラムが，３つの処理の単位，ｐｇ１，ｐｇ２，ｐｇ３，に分割されている．ｐｇ１，ｐｇ２などをプログラムグレインと呼ぶ．プログラムグラフの分割のパタンは一種類とは限らず，他の分割も可能である．

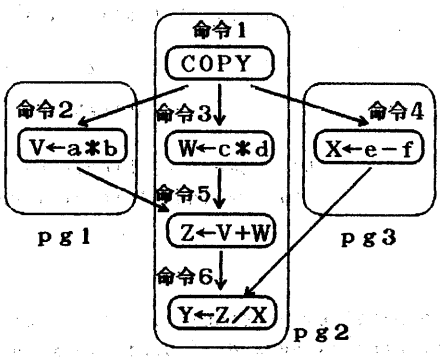

図2.2 図2.1をＣで分割した結果

プログラムグレインに関する仮定は以下の通りである．

仮定 G.1)
プログラムグレインに含まれている命令は，すべて逐次的に実行順序が決められ，プロセッサによって実行される．

仮定 G.2)
プログラムグレインの処理が開始されると，命令は，決められた順序に従い，各命令一回ずつ，一つ残らず逐次に実行される．
（この仮定は，プログラムグレインの実行制御を簡単にするために設けられている．）

ここで，図2.3のように，異なるプログラムグレインｐｇＡ，ｐｇＢが１つ以上の命令を共有した場合を考える．命令３は，命令１からの通信と命令２からの通信が行われた後に実行される．もしも，ｐｇＡに含まれる命令３，命令４が実行されたとすると，ｐｇＢに含まれる命令３，命令４が実行されずに残ってしまう．このことは仮定G.2に矛盾する．したがって，次の定理が導かれる．

定理 G.1)
異なるプログラムグレインは，命令を共有しない．

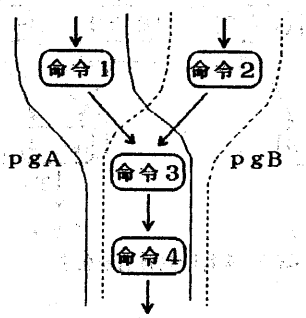

図2.3 異なるプログラムグレイン間の命令の共有

2.3 ハードウェアに関する仮定

プロセッサに関する仮定は以下の通りである．

仮定 P.1)
各プロセッサには，同じプログラムが格納されている．

仮定 P.2)
各プロセッサはすべて同じ性能を持つ．

仮定 P.3)
プロセッサは，一度に一つの命令しか実行できない．したがって，プロセッサは命令を逐次的にしか実行できない．

仮定 P.4)
プロセッサには，複数のプログラムグレインを割り当てることができ，それらの処理は逐次的に行われる．

通信に関する仮定は次の通りである．

仮定 C.1)
通信には，外部通信と内部通信がある．

仮定 C.2)
外部通信は，プロセッサ間の通信である．この通信はネットワークを介して行われ，一般に，通信時間がかかる．ここでは，その通信にかかる時間を一定とする．

仮定 C.3)
内部通信は，プロセッサ内部で行われる通信である．プロセッサ内部の通信は，プログラムカウンタを用いて行う．したがって，プログラムグレイン内部の通信や同一のプロセッサに割り当てられたプログラムグレイン間の通信は高速に行われ，その通信時間を０とみなす．

3. プログラムグラフの分割法

プログラムグラフは，以下に挙げる分割法によって分割される．

3.1 基本的な処理の分割法

(1) 逐次化分割法

逐次化分割法では，プログラムグラフをまったく分割せず，一つのプログラムグレインと考える．
この分割法を用いた場合，仮定G.2から，与えられたプログラムグラフ中の命令は，すべて逐次的に実行されることになる．また，各命令間の通信はすべてプログラムグレイン内部の通信となる．これは，従来のノイマン処理である．

(2) 完全分割法

完全分割法は，プログラムグラフに存在するすべての通信箇所で分割する方法である．したがって，与えられたプログラムグラフにｎ個の命令がある場合，与えられたプログラムグラフからｎ個のプログラムグレインができる．この分割法を用いた場合，各命令間の通信はすべてプログラムグレイン間の通信になる．
これらのプログラムグレインを，各々１台のプロセッサに割り当てて実行させる．このような分割法と割当法ではプロセッサ間の通信が必要以上に増えてしまうので，プロセッサ間のネットワークのトラフィックが増大し，処理の効率が低下する可能性がある．

(3) 基本分割法

基本分割法は，実行を開始するのに複数の通信が必要な命令の直前で分割する方法である．プログラムグラフをこの方法で分割してできる

プログラムグレインを，各々１台のプロセッサに割り当てて実行させる．

あるプログラムグラフの一部を図3.1のように分割して，各々１台のプロセッサに割り当てて実行させることにする．この場合，命令１から命令２への通信が行われた時に，まだ命令４から命令２への通信が行われていなかったとすると，その通信が行われるまでプロセッサは命令２を実行することを待つことになる．プロセッサはこの待ち時間の間，他にするべき事があったとしても，このプログラムグレインの実行のために占有され続けることになり，プロセッサは無駄な時間を使ってしまう．基本分割法は，このような無駄な待ち時間が生じないように分割する分割法である（図3.2）．基本分割法を用いて分割すれば，命令１の実行を終了してその終了を示す信号を送り終わったプロセッサは解放され他のプログラムグレインの実行を始めることが可能になる．命令２を含むプログラムグレインを実行するプロセッサは，命令１と命令４からの両方からの通信を受け取った時に，命令２の実行を開始する．

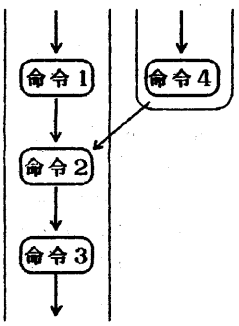

図3.1　無駄な待ち時間を生ずる分割例

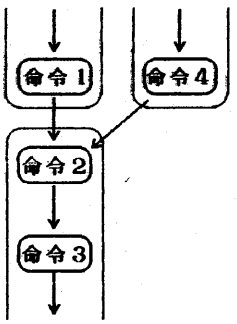

図3.2　基本分割法

(4) 実行時刻算出分割法

図3.2の基本分割法において，もしも命令１から命令２への通信が行われた時に，既に命令４から命令２への通信が行われてしまっていたとすると，プロセッサは待ち時間なしに命令２を実行することができる．また，プログラムグレイン間の通信も一つ減らすこともできる．したがって，このような場合には，図3.1のように分割する方がよいと思われる．しかし，基本分割法においては，各命令の実行開始時刻や実行終了時刻が算出できないとしたので，図3.1のように分割することができなかった．

実行時刻算出分割法では，プログラムグラフの各命令の実行開始時刻や実行終了時刻を算出して，基本分割法で得られたいくつかのプログラムグレインを一つのプログラムグレインにまとめる．この分割法を使うと，プログラムグレイン間の通信を減らすことができる．プログラムグラフをこの方法で分割してできるプログラムグレインを，各々１台のプロセッサに割り当てて実行させる．

各命令の実行開始時刻と実行終了時刻の算出の方法と分割の方法を，図3.3を用いて説明する．

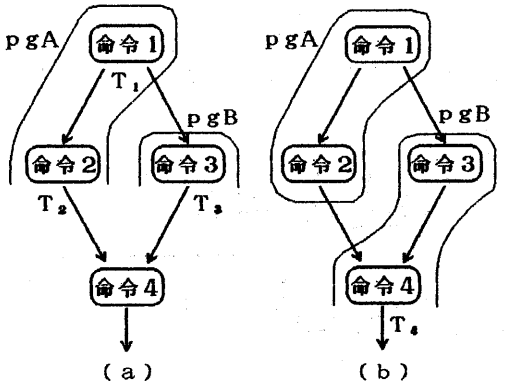

図3.3　実行時刻算出分割法

まず始めに，プログラムグラフに含まれている命令のうち一番最初に実行される命令（一般には一つと限らない），すなわち命令１の実行開始時刻を０とする．また，命令ｉの実行に必要な時間をe_iとする．すると命令１の実行終了時刻T_1はe_1となる．命令１を含むプログラムグレインをｐｇＡとする．命令２，命令３のどちらかをｐｇＡに含めてもよいが，ここは左側を優先して，命令２を含めることにする．そして命令３を含むプログラムグレインをｐｇＢとする．プログラムグレインの中の通信にかかる時間を０と仮定したので，ｐｇＡにおける命令２の実行開始時刻はT_1となり，その実行終了時刻T_2は$T_1 + e_2$となる．また，プログラムグレイン間の通信にかかる時間をｃとすると，ｐｇＢにおける命令３の実行開始時刻は$T_1 + c$となり，その実行終了時刻T_3は$T_1 + c + e_3$となる．

ここで，命令４をｐｇＡ，ｐｇＢのどちらに含めるのかが問題になるが，実行時刻算出分割法では，命令２，命令３の各実行終了時刻T_2，T_3のうち，実行終了時刻が遅い方の命令を含んでいるプログラムグレインの方に命令４を含める．すなわち，図3.3の(a)において，$T_2 < T_3$（T_2よりもT_3の方が遅い）とすると，図3.3の(b)に示すように，命令４はｐｇＢに含められる．

次に，命令４の実行開始時刻と実行終了時刻を求めてみる．命令２から命令４への通信は，プログラムグレイン間の通信であるから時間ｃだけかかり，命令４への通信が行われる時刻は$T_2 + c$となる．また，命令３から命令４への通

信は，プログラムグレイン内の通信であるから時間０で行うことができ，命令４への通信が行われる時刻はT_3となる．したがって，命令４の実行開始時刻は，max（T_2+c，T_3）となる．さらに，実行終了時刻T_4は，max（T_2+c，T_3）＋e_4となる．

以上の事を繰り返すことによって，実行時刻算出分割法は，各命令の実行開始時刻と実行終了時刻を算出しながらプログラムグラフを分割してゆく．

実行時刻算出分割法においては，プロセッサに待ち時間が生じる場合がある．

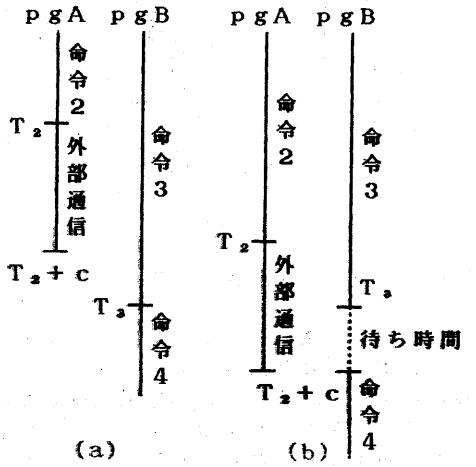

図3.4　実行時刻算出分割法において，
プロセッサに待ち時間が生じる例

図3.4は，図3.3の(b)のように分割されている場合に，プロセッサに待ち時間が生ずる例を示したものである．図3.4の(a)(b)において，$T_2<T_3$であるとする．図3.4の(a)は，T_2よりもT_3の方が充分に遅い場合（$c \leqq T_3-T_2$）である．この場合，pgBにおいて命令３から命令４に内部通信が行われた時には，既に命令２から命令４に外部通信が行われてしまっているため，命令３の後，待ち時間なしで命令４が実行される．一方，図3.4の(b)は，T_2よりもT_3が少しだけ遅い場合（$0<T_3-T_2<c$）である．この場合，pgBにおいて命令３から命令４に内部通信が行われた時に，まだ命令２から命令４に外部通信が行われていないため，プロセッサはその通信が行われるまで，時間（T_2+c）$-T_3$だけ，命令４の実行を待たされてしまう．pgBが割り当てられているプロセッサはこの待ち時間の間，たとえ実行可能な命令が他にあったとしても，それを実行することはできない．しかし，この待ち時間は無制限に大きくはならない．待ち時間は，

$$(T_2+c)-T_3=c-(T_3-T_2)<c$$

となり，つねに１外部通信時間よりも短くなる．

3.2 条件分岐処理の分割法

条件分岐処理は，図3.5のような条件分岐命令と図3.6のような２種類のゲート命令によって実現されている．図3.5に示した条件分岐命令は，条件$x>y$が成立するときｚに値TRUEを代入し，成立しないときｚに値FALSEを代入する，という命令である．図3.6に示したＴゲート命令は，参照すべき変数ｚの内容がTRUEの時は次に続く処理を行い，FALSEの時は次に続く処理を行わない，という命令である．これとは逆に，Ｆゲート命令は，参照すべき変数ｚの内容がFALSEの時は次に続く処理を行い，TRUEの時は次に続く処理を行わない，という命令である．

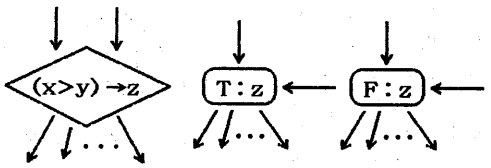

図3.5　条件分岐命令　　図3.6　ゲート命令

条件分岐処理は，これらの命令を使って，図3.7のように実現される．すなわち，条件$x>y$が成立するとき，処理Ａを行い，処理Ｂを行わない．逆に，条件が成立しないとき，処理Ａを行わず，処理Ｂを行う．

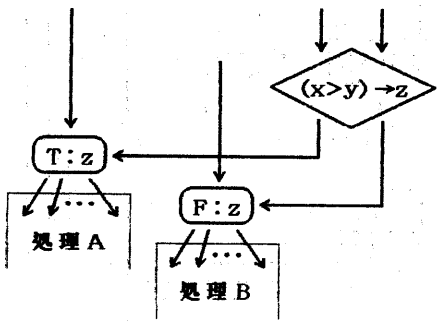

図3.7　条件分岐処理

一般に，条件分岐処理では，条件の成立・不成立は，静的には非決定的である．したがって，ゲート命令に続く処理の実行は非決定的になる．もしも，図3.8のように分割すると，条件がFALSEの場合に，Ｔゲート命令に続く命令が実行されずに残ってしまう．このことは仮定G.2に矛盾するから，図3.9のように，Ｔゲート命令の直後

で分割する．Fゲート命令についてもまったく同様である．

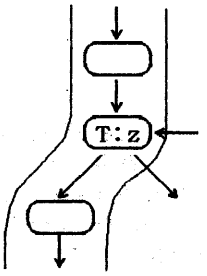

図3.8 矛盾のある分割

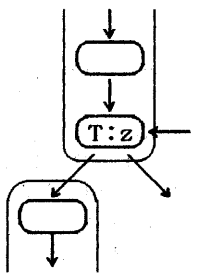

図3.9 矛盾のない分割

3.3 ループ処理の分割法

ループ処理は，図3.10のようなマージ命令と先に述べた条件分岐命令・ゲート命令によって実現されている．図3.10に示したマージ命令は，処理A・処理Bの少なくともどちらか一方が終了した後，マージ命令の後に続く処理cを行う，という命令である．

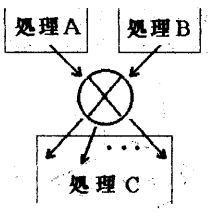

図3.10 マージ命令

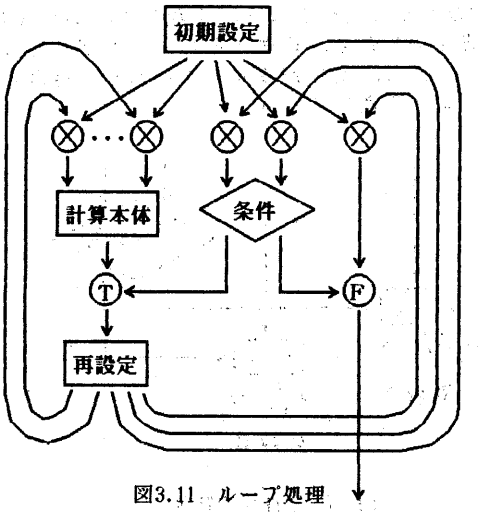

図3.11 ループ処理

ループ処理は，これらの命令を使って，図3.11のように実現される．まず始めに，ループに入るための初期設定を行う．次に，マージ命令を通ってループにはいる．ループの中では，その中で行われるべき計算の本体と，ループを脱出するかどうかの条件判断が並列に行われる．もしも，ループの条件部が真になれば，各変数の再設定を行い，再度ループにはいる．ループの条件部が偽になれば，ループを脱出する．

ループ処理において，もしも，図3.12のように分割すると，右からの入力によってマージ命令が実行される場合に，マージ命令よりも前の命令が実行されずに残ってしまう．このことは仮定G.2に矛盾するから，図3.13のように，マージ命令の直前で分割する．

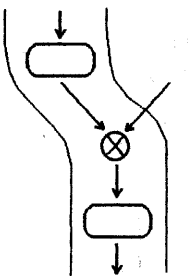

図3.12 矛盾のある分割

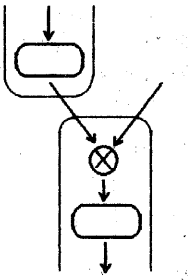

図3.13 矛盾のない分割

3.4 プロシジャコールを含む処理の分割法

プロシジャは，図3.14のようなプロシジャコール命令によって呼び出されて実行される．図からわかるように，プロシジャに関する処理は，本質的にプロシジャコール命令を，プロシジャ本体でおきかえる処理と考えることができる．

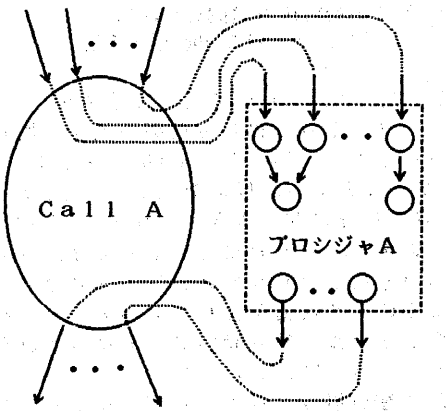

図3.14 プロシジャコール命令

いま，プロシジャAが2箇所の異なる場所から呼び出されているとして，この場合のプロシジャコール命令をプロシジャ本体でおきかえると，図3.15のようになる．これを今までの分割法で分割すると，同図(a)と(b)に示すようなプログラムグレインpgAとpgBができ，1つのプロシジャが，呼び出し側のプログラムの違いで，別のプログラムグレインに分けられることになる．このようなことは，プロシジャから

戻るときにもおこる．このことは，プロシジャコールによって，プログラムの分割が影響を受けることを表しており，また，プロシジャコールがネスティングしているときには，そのネスティングの深さ方向にプログラムを解析しなければならないことを表している．これらの影響の解析は非常に複雑になることが予想され，これらを行うことは一般に実用的ではないと考えられるので，本論文では，プロシジャコール命令の直前および直後で分割する．

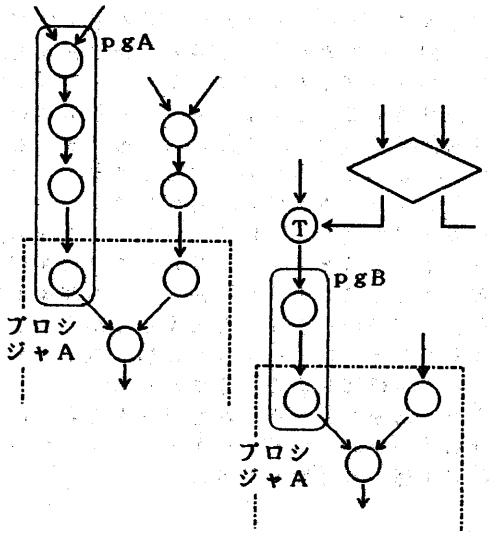

図3.15 プロシジャAが2箇所の異なる場所から呼び出されている例

4．評価の方法

今回の評価に採用した問題は，２つである．一つは，条件分岐処理やループ処理などを含まない基本的な処理だけで構成されるプログラムグラフである．この代表として，簡単な構造体処理（５つの要素から成る構造体の１つの要素を書換えるプログラム）を扱う問題を採用した．プログラムグラフ自身が持つ命令実行に関する平均並列度は，2.50 である．

もう一つは，ループ処理だけで構成されるプログラムグラフ（フィボナッチ数を求めるプログラム）である．ループ処理だけで構成されるプログラムグラフの処理時間の比較は，繰り返し回数が充分に大きいとして，ループへの入力，ループからの出力にかかる処理時間を無視し，ループの計算本体の処理時間で比較する．このプログラムグラフのループの計算本体自身が持つ命令実行に関する平均並列度は，2.58 である．

上の各々の問題を，本研究室で開発された並列処理用言語Ｐ[3]で記述し，そのコンパイラによってプログラムグラフを得る．これを，これまでに述べてきた４つの分割法，逐次化分割法，完全分割法，基本分割法，実行時刻算出分割法，で分割するプログラムを用いてプログラムグレインを得る．そして，それぞれのプログラムグレインを各々１台のプロセッサに割り当てて実行させるとする．それぞれの場合において，処理時間が外部通信時間にどれだけ影響を受けるかを調べるために，実行時の外部通信時間をパラメータとして処理時間を計算し，グラフ化して比較する．

5．結果と考察

図5.1は構造体処理を行うプログラムグラフについて評価した時の結果，図5.2はフィボナッチ数を求めるプログラムグラフについて評価した時の結果をグラフ化したものである．

縦軸は，処理時間を表し，横軸は，相対通信時間を表している．ただし，相対通信時間は，次のように定義されている．

$$\text{相対通信時間} = \frac{\text{外部通信時間}}{\text{1命令実行時間の平均}}$$

また，相対通信時間は，コンピュータのアーキテクチャに依存するパラメータと見なすこともできる．すなわち，これが大きいということは，外部通信に時間がかかるアーキテクチャを表している．

逐次化分割法は外部通信がまったくない逐次処理なので，一定値になっている．これは，ノイマン処理と考えることができるので，他の並列処理にとって一つの重要な基準となる．

グラフの傾きは，処理時間が外部通信時間にどれだけ影響を受けるかを表している．その傾きが大きいほど，外部通信時間に影響を受けやすいことになる．完全分割法や基本分割法に比べて，実行時刻算出分割法の方は，傾きがかなり小さい．したがって，外部通信時間に影響をかなり受けにくいことがわかる．

完全分割法は，相対通信時間が約1.5以下になるアーキテクチャに対して，逐次化分割法よりも処理時間が短くなる．また，基本分割法は，相対通信時間が約2.0以下になるアーキテクチャに対して，逐次化分割ほうよりも処理時間が短くなる．これらに対し，実行時刻算出分割法は，相対通信時間が約4.3以下になるアーキテクチャに対して，逐次化分割法よりも処理時間が短くなる．したがって，完全分割法や基本分割法に比べ，実行時刻算出分割法は，より多くのアーキテクチャに対して，逐次化分割法よりも処理時間が短くなる．

また，アーキテクチャを固定した場合，実行時刻算出分割法を用いて実行させたときの処理時間が，完全分割法・基本分割法を用いて実行させたときの処理時間よりも短いことがわかる．

これらの結果は，完全分割法や基本分割法は，実際に分割しなくてもよいところでも分割して外部通信を必要以上に増やしているのに対し，実行時刻算出分割法は，命令が実行される時刻を算出することによって，分割しなければならない所だけで分割して外部通信をできるだけ減らしていることが原因と思われる．

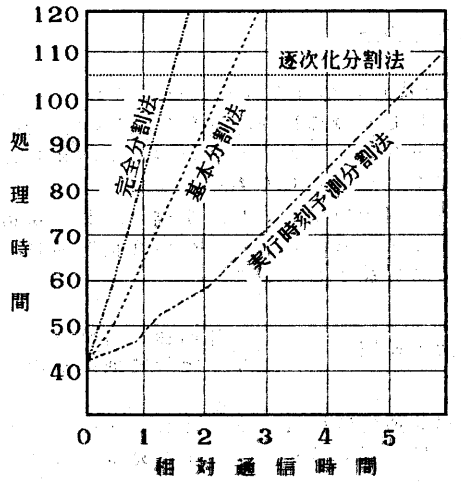

図5.1 構造体処理を行うプログラム

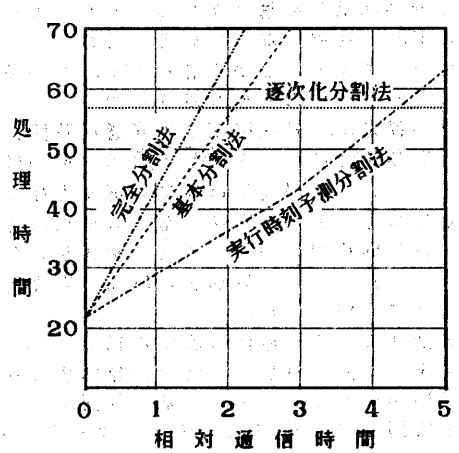

図5.2 フィボナッチ数を求めるプログラム

図5 相対通信時間とと処理時間の関係

6. むすび

以上，並列コンピュータのプログラムグラフの分割について考察を行った．

一般に各命令の実行時刻を正確に算出することは非常に困難である．命令実行時刻の算出法は，非常に簡単なものであったにもかかわらず，外部通信時間の処理時間への影響を，基本分割法の約３分の１にすることができる．今後，いろいろな処理を含むプログラムグラフについても評価しながら，より適切な命令実行時刻の算出ができるように改良すれば，外部通信時間の処理時間への影響がさらに減る可能性がある．

多くのプログラムグレインを１台のプロセッサに割り当てることによって，それらの間の通信は内部通信になり，外部通信を減らすことができる．しかし，１台のプロセッサは，プログラムグレインを逐次にしか処理できないから，１台のプロセッサにプログラムグレインを多く割り当てすぎると，プログラムグラフの並列度を落とすことになる．したがって，外部通信時間と，並列度のバランスを考慮しながら，プログラムグレインをプロセッサに割り当てる方法についても研究を進める必要がある．

また，結果として得られたデータをもとに，各分割法が有効となるアーキテクチャについても研究を進める必要がある．

［参考文献］

[1] 伊藤，曽和：並列処理コンピュータを構成するためのプロセスグレインに関する基礎考察，電気関係学会東海支部連合大会，1988.

[2] 伊藤：並列処理コンピュータを構成するためのプロセスグレインに関する研究，名古屋工業大学卒業論文，1988.

[3] 奥平，曽和：データフロー用言語P，データフローワークショップ1987，pp.271-277，1987.

[4] Kruatrachue,B. and Lewis,T. : Grain Size Determination for Parallel Processing,Software，Vol.5，No.1，pp.23-32，1988.