計算機アーキテクチャ 59−3
オペレーティング・システム 29−3
(1985・10・24)

プログラム可能論理演算装置の処理効率について

The Efficiency of a Programmable Logic Unit

村山 隆彦[+] 山田 秀和[+] 吉岡 良雄[++] 中村 維男[+] 重井 芳治[+]

Takahiko MURAYAMA Hidekazu YAMADA Yoshio YOSHIOKA Tadao NAKAMURA Yoshiharu SHIGEI

[+] 東北大学工学部 [++] 岩手大学工学部

Tohoku Univ. Iwate Univ.

1．はじめに

フォン・ノイマンがプログラム内蔵方式の計算機を提言して以来、計算機の発達にはめざましいものがある。デバイス面では、集積回路（ＩＣ）、大規模集積回路（ＬＳＩ）を経て、現在は超ＬＳＩの時代である。さらにＧａＡｓ、ＨＥＭＴ、ジョセフソン素子などの高速デバイスの研究が各所で行われている。この間、スループットの向上を目的として、チャネル、仮想記憶、キャッシュ、パイプライン処理の導入など、アーキテクチャ面の改善も行われてきた。

しかしながら、現在のノイマン型計算機は、次の三つの問題点を持っている。一つめは、真に処理の対象となるデータ語以外の多くの情報、すなわち、命令やアドレス生成情報が一本のパスを流れる点であり、フォン・ノイマン・ボトルネックと呼ばれている[1]。二つめは、単一のプログラム・カウンタによって、処理のシーケンスが規定される点である。処理されるアルゴリズムは、単一フロー制御でなければならない。すなわち、命令レベルでの並列実行可能性を基本的に持たない。三つめは、プログラムとデータが同じメモリ領域に格納されている点である。

このような問題点を克服するために、データフロー計算機、リダクション計算機などのいわゆる非ノイマン型計算機と呼ばれるものが各所で研究・開発されている[2],[3]。その中の一つとして、高速処理を目的とする計算機アーキテクチャのための装置－プログラム可能論理演算装置（ＰＬＵ：Programmable Logic Unit）が提案されている[4]。この装置の特徴は、中央処理装置（ＣＰＵ：Central Processing Unit）の演算機能をメモリ上に分散配置し、ハードウェア上にその演算のための専用回路を動的に構築して、データの流れだけによって、ゲート遅延時間レベルで演算が可能な点である。

本論文では、ＰＬＵを用いた計算機について、その基本動作について述べた後、ＰＬＵと制御装置（ＣＵ：Control Unit）の同期に絡み、ＣＵの待ち時間を有効に活用する手段として、ＰＬＵの多重化を考える。そして処理の並列性を考慮し、効率、有効性などについて、検討する。

2．ＰＬＵについて

本章では、ＰＬＵの一般的特徴について示した後、試作機を例に、その構造について述べる。さらに、ＰＬＵを用いた計算機システムを考え、その演算処理の方法、及びＰＬＵとＣＵの同期問題について考察する。

2.1 ＰＬＵの一般的特徴

ＰＬＵは、ハードウェア上に演算プログラムを格納することによって専用回路が構成され、その後、演算データを格納するだけで演算結果

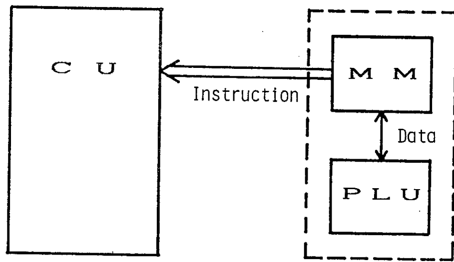

図1．ＰＬＵを用いた計算機の構成図

が得られる装置である。演算の実行は、外部からみればＰＬＵとのデータの転送であり、またＰＬＵ内部では垂れ流し的に処理が進む。よって、フォン・ノイマン・ボトルネックの緩和、及び高速演算が期待できる。さらに"演算"を完全に制御装置（ＣＵ：Control Unit）から切り離してしまったため、ＣＵはその名の通り、フロー制御に専念できる。

2.2 ＰＬＵの構成

ＰＬＵは、命令や演算データを格納するレジスタ、及び演算を行うＡＬＵからなる処理要素（ＰＥ：Processing Unit）を基本単位とし、各ＰＥが簡単なネットワーク状に結合されたものである。

外側からは、ＰＥの演算機能、接続形態を規定する命令レジスタ（ＩＲ：Instruction Register)、及び演算データの格納、演算結果の出力のためのインタフェースとなる結果レジスタ（ＲＲ：Result Register）のみアクセスすることができる。したがって、従来主記憶（ＭＭ：Main Memory）に格納されていた演算命令がＩＲ用データに、演算データ(の一部)がＲＲ用データにそれぞれ対応し、各ＰＥにアドレスを割り付けることによって、その内部にデータの記憶と演算の機能を併せ持つようなメモリの一部として考えることが出来る（図1)。

ＰＬＵの構成法としては種々考えられるが[5]、ここではその一例として、著者らが試作したＰＬＵについて述べる。図2に試作ＰＬＵ１ステップ（ＰＥ）の構成図を示す。試作機ではデータの他にフラグも伝播させ、簡単な条件判断もできるように設計されている。また、

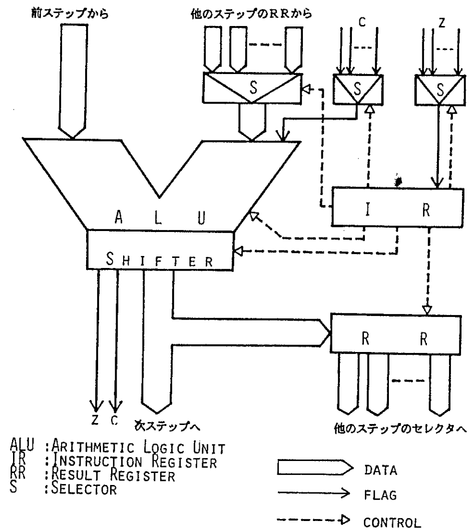

図2．試作ＰＬＵ１ステップの構成

データの依存性が大きいと思われるため、ＡＬＵの入力の一方は直前のステップと接続され、他方はセレクタを介して専用の接続線により、前ステップのいずれかと接続するという形態をとっている。しかし、ＰＬＵの容量が大きな場合、限られた接続線では汎用性に欠けるため、複数本の共通バス（分割バス）によってステップ間を接続する方法も提案されている[6]。

2.3 ＰＬＵにおける演算処理

一般の処理プログラムは、演算を実行する部分と、制御の流れを変える部分に分けられる。ＰＬＵは、演算をプログラム可能な専用回路によって高速化し、処理全体の性能改善を図ろうとするものである。ＰＬＵでの演算は、データの"垂れ流し"によって、流れにそって処理され、数式に潜在する並列性を引き出すことも可能である。したがって、できる限り制御部分を小さくするようなマッピング法が望まれる。

本節では、前述のＰＥをタンデムに接続したＰＬＵを例に、その演算における性質を述べる。

ＰＬＵにおける演算処理は、以下のようにして実行される。最初に、演算プログラム（ＩＲ用データ）をＩＲに格納して、所望の演算回路を構築（マッピング）する。その後、演算デー

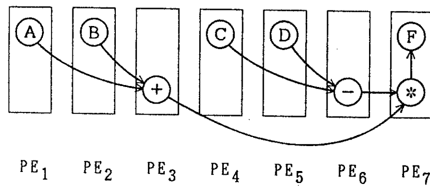

図3．数式のマッピング例

タ（ＲＲ用データ）を対応するＲＲにそれぞれ格納すると、あるゲート遅延時間後に別のＲＲに演算結果が得られる。

演算プログラムのマッピングはかなりのオーバヘッドとなるため、ＰＬＵの容量が十分大きければ、コンパイル時、若しくはロード時に直接構築するのが望ましい。ＰＬＵの容量が小さいときには、仮想化の技術を用いる方法も提案されている[7]。

また、逆ポーランド記法で記述された数式をそのままＰＬＵへマッピングすることができ（図３）、コンパイラの負荷が軽減できる。

2.4 ＰＬＵとＣＵの同期

データの"垂れ流し"で演算を実行するＰＬＵと、同期クロックで動作するＣＵとは互いに独立に処理を行い、高速演算が期待できる反面、データ転送などの際には何等かの同期をとる必要がある。ＰＬＵにおける演算時間は、その演算・データによって広く分布しており、必要とするデータ（演算結果）が有効か否かを判断しなければならない。そこで本節ではＰＬＵとＣＵの同期方法について概説する。

Ｉ．ソフトウェアによる同期

ソフトウェアによる同期法は、ハードウェア・コストを低く押さえることができる反面、コンパイラ等の負荷を増大させる恐れがある。

まず、コンパイラによって入出力ノード間の距離を計算し、NOP命令などを挿入して結果が得られるのを待つ、という方法がある。この場合、遅延のばらつきにかかわらず誤動作を起こさないように、各ＰＥにおける遅延は最大の場合を考えなければならない。その間、ＣＵは待ち状態となり、効率が低下する。

この待ち状態を有効に利用するために、ＰＬＵを多重化してＰＬＵの演算時間を相対的に小さくし、ＣＵの制御時間の陰に完全に隠してしまう、という方法も考えられる。この場合、スループットは向上するが、ターンアラウンドタイムは低下する。

II．ハードウェアによる同期

ハードウェアによる同期法は、何等かの付加ハードウェアを必要とするが、高速化が期待できる。付加ハードウェアは、ＰＥ間の通信に使用されるものと、ＰＥ内の完了信号として使用されるものに大別できる。

ｉ）ＰＥ間の同期（通信）

ＰＥ間で同期をとることによって、データの存在が明らかとなり、パイプライン的に順々にデータを送り込むことが可能となる。具体的には、ＡＣＫ信号を用いる方法、トークンを用いる方法、二相クロックを用いる方法（同期通信方式）などが考えられる。

ii）ＰＥ内の同期（完了信号）

ＰＥ内にハードウェアを付加し、完了信号を発生させる。簡単な方法として、遅延素子が挙げられるが、最大遅延で見積もるため、無駄な待ち時間が生じる。正確な完了信号が得られる方法として、二線式論理が挙げられるが[8]、ハードウェア量の増加は免れない。

３．ＰＬＵの処理効率について

ＰＬＵは、"垂れ流し"的なデータの流れによって処理が進み、これをＣＵによって制御の流れを変えたり、プログラムのマッピングを行ったりして、一連の処理プログラムが実行される。したがって、ＰＬＵとＣＵの処理速度の差が処理プログラム全体の実行時間を大きく左右する。本章ではＰＬＵを用いた計算機について、基本演算レベル、タスクレベル、ジョブレベルでの並列性を考慮し、ＰＬＵの多重化という概念を導入して、その効率、有効性などについて検討する。

3.1 数式処理

一般のノイマン型プログラムをPLUで処理する場合の実行時間について考える。

PLUを用いた計算機における数式の実行は、一般にMM－PLU間の演算データの転送である。したがって、機械語レベルでみたPLUを用いた計算機の命令体系は、従来のノイマン型計算機の命令体系とはかなり異なったものとなる。本節では、二入力一出力の演算の集合、すなわち二進木で表現できる演算の処理について、その実行時間を検討する。

PLUでは、従来メモリに格納され、逐次的に実行されていた低レベルの演算命令の集合がデータの依存関係によって結合され、多入力多出力の高レベルの演算命令に置き換えられたものと考えることができる。

ここでは、PLUの容量は十分大きく、コンパイル時、若しくはロード時に所望の回路がPLU上に構築されているものとする。

PLUにおける演算時間は、MM－PLU間のデータ転送時間 t_{mp}、及びPLU内のゲート遅延時間 t_{gd} に大別される。PLU内部は"垂れ流し"的に実行されるため、データが転送されると同時に演算が開始される。したがって、転送と演算は一部並行して行われる。

一般には $t_{mp} >> t_{gd}$ であるため、個々の演算に要する時間はデータの転送に要する時間に隠れる。したがって、演算子nの数式に対する演算時間 $t_{plu}(n)$ は、

$$t_{plu}(n) = (n+2) \cdot t_{mp} + t_{gd} \qquad (1)$$

となる。ただし、t_{gd} は最後の入力データが転送されてから出力結果が確定するまでのゲート遅延時間である。

t_{gd} を推定するために各PEにおける演算時間分布は等しいと仮定し、$g(\tau)$ とする。このとき、最後の入力PEから出力PEまでのPE数をpとすると t_{gd} は、

$$t_{gd} = \int_0^{\tau} \tau \cdot g_p(\tau) d\tau \qquad (2)$$

で与えられる。ただし、$g_p(\tau)$ は $g(\tau)$ のp回の畳み込みとする。いま、二進木で表される数式を考えているから、演算ステップ数pの範囲は数式の並列度（並列に実行できる演算の合計）を $f_{para}(n)$ として、

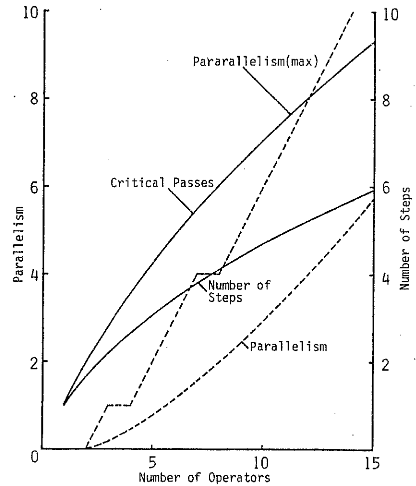

図4．数式の並列度と演算ステップ数

$$1 \leq p \leq n - f_{para}(n) \qquad (3)$$

で表される。あらゆる形の二進木について、その並列度と演算ステップ数を調べたものを図4に示す。演算子の数が増加しても数式内に潜在する並列性によって演算ステップ数はそれほど増加しない。もし、理想的な転送が行われて、入力データがすべて同時刻に揃えば、演算時間は演算ステップ数の最大値 p_{max} で定まることになり、並列性を最も多く持っている完全二進木の型の数式の場合には、その並列度は、$(n - \lceil \log_2(n+1) \rceil)$、クリティカルパス (p_{max}) は、$\lceil \log_2(n+1) \rceil$ で与えられる。ただし、「 」は天井関数を表す。

3.2 繰り返し処理

PLUは、任意の演算のための専用回路をその内部に構築して、MMとのデータ転送のみによって、ゲート遅延時間レベルで演算を行う。したがって、理想的には与えられたソースプログラムをすべて展開し、PLU上にそのプログラムのための専用処理装置を構築するのが望ましい。

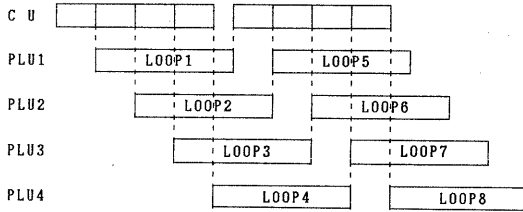

図5．繰り返し演算の実行

しかしながら、テンプレート製作のための時間・空間的オーバヘッドも大きい。特にループなどの繰り返し処理において、制御変数が動的に決まる場合、若しくは繰り返し回数が膨大な場合にはあまり現実的ではない。

そこで本節では、ループ内のタスクの並列性を考慮し、繰り返し処理における実行時間を検討する。加えて、パイプライン方式との比較を行う。

まず、ループ内で実行される処理がマッピングされているテンプレートを用いて、繰り返しの回数だけ順々に処理を実行していく場合、すなわちループを全く展開しない場合を考える。このときの実行時間 $T_{plu}(1)$ は、τ_1 を単位時間、$s_1 \cdot \tau_1$ をアドレス計算などのオーバヘッド時間、$c_1 \cdot \tau_1$ をループのフロー制御時間、d_1 をPLUにおける処理時間（ゲート遅延時間）、nを繰り返し回数とすると、

$$T_{plu}(1) = (s_1 + n \cdot c_1) \cdot \tau_1 + n \cdot d_1 \qquad (4)$$

で与えられる。

前節でも触れたように、一般には $c_1 \cdot \tau_1 >> d_1$ であるが、配列データなどの場合、ベクトルレジスタの設置、メモリインタリーブなどの手法により、$c_1 \cdot \tau_1$ をかなり小さくすることができる。さらに、ループ内の処理が多い場合、PLUにおける処理の間、CUは待ち状態となり、相対的に効率が低下する。そこで、すでにマッピングされているテンプレートのコピーを動的に作りだし、ループをある程度展開して、PLUにおける処理をタスクレベルで並列に実行することを考える（図5）。このときの実行時間 $T_{plu}(N)$ は、

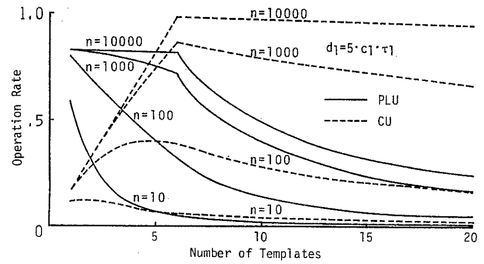

図6．PLUとCUの稼働率

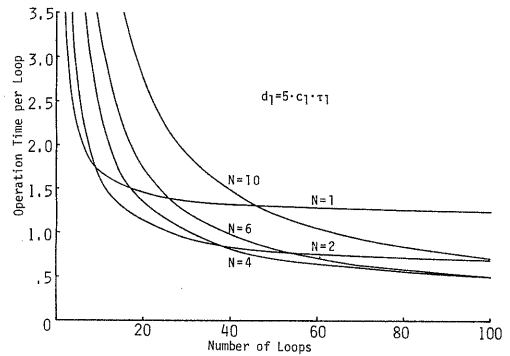

図7．実行時間（その1）

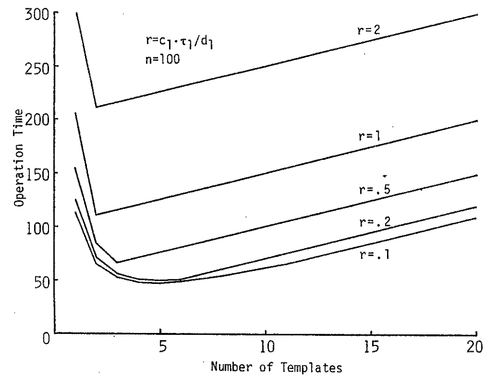

図8．実行時間（その2）

$$T_{plu}(N) = \begin{cases} (s_1 + (n - (\lceil n/N \rceil - 1) \cdot (N-1)) \cdot c_1 \\ \quad + (N-1) \cdot I_1) \cdot \tau_1 + \lceil n/N \rceil \cdot d_1 & ((N-1) \cdot c_1 \cdot \tau_1 > d_1) \\ (s_1 + n \cdot c_1 + (N-1) \cdot I_1) \cdot \tau_1 + d_1 & ((N-1) \cdot c_1 \cdot \tau_1 \leqq d_1) \end{cases} \qquad (5)$$

で与えられる。ただし、Nは展開したテンプレートの数、$I_1 \cdot \tau_1$ はテンプレート製作のためのオーバヘッド時間とする。

d_lを単位時間、$s_l \cdot \tau_l = I_l \cdot \tau_l = 5 \cdot d_l$として、$c_l \cdot \tau_l$（制御時間）と$d_l$（演算時間）の比、n、及びNをパラメータとした場合のPLU（1テンプレート当たり）とCUの稼働率を図6に、処理時間を図7,8に示す。多重化によって、資源の有効利用がみられる。処理時間については、nが小さければ、テンプレート製作のオーバヘッドのため、効果がほとんどない。しかし、nが大きくなると、その時間が相対的に小さくなり、有効性が得られる。また、制御時間と演算時間の比によって、最適な多重度Nを持つ。

ここで、$\lceil n/N \rceil = n/N + 1/2$ と近似して、処理時間を最小にするNを理論的に求める。式(5)の第一式は、

$$T_{plu}(N) = (s_l + (n-(n/N-1/2) \cdot (N-1)) \cdot c_l + (N-1) \cdot I_l) \cdot \tau_l + (n/N+1/2) \cdot d_l \quad (6)$$

となり、これをNで微分して最適なN_{opt}を求めると、

$$N_{opt} = 2 \cdot (c_l + d_l/\tau_l) \cdot n/(c_l + 2 \cdot I_l) \quad (7)$$

を得る。一方、式(5)の第二式から、

$$N_{opt} = d_l/c_l \cdot \tau_l + 1 \quad (8)$$

を得る。したがって、PLUを用いた計算機における繰り返し演算の最大スループット$Th_{max}(n)$は、

$$Th_{max}(n) = \begin{cases} n/(s_l + (c_l + d_l) \cdot n + d_l/2) & ((N-1) \cdot c_l \cdot \tau_l > d_l) \\ n/(s_l + n \cdot c_l + d_l \cdot I_l/c_l + d_l) & ((N-1) \cdot c_l \cdot \tau_l \geq d_l) \end{cases} \quad (9)$$

となる。

比較対象として、パイプライン方式による演算を考える。簡単化のため、ループ内の処理は一本のパイプラインで実行可能とする。s_iをアドレス計算などのオーバヘッド時間、l_iをセグメント数、τ_iをパイプラインのピッチとすると、パイプライン方式による繰り返し演算のスループット$Th_{pipe}(n)$は、

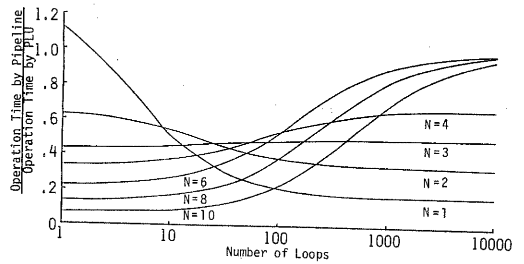

図9．PLU方式とパイプライン方式の比較

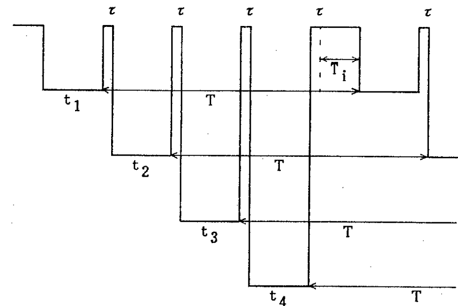

図10．多重プログラミング

$$Th_{pipe}(n) = n/((s_i + l_i + (n-1)) \cdot \tau_i) \quad (10)$$

で与えられる。

$s_l \cdot \tau_l = s_i \cdot \tau_i$、$2 \cdot d_l = l_i \cdot \tau_i$としたときのPLU方式とパイプライン方式のスループットの比較を図9に示す。直線状に単調増加するパイプライン方式の実行時間と比較して、PLU方式の場合、Nの値が小さければ、加速率が大きく、Nの値がある程度以上大きければ、繰り返し回数の増加にしたがってスループットも向上する。しかし、タスクレベルでの並列性が$d_l / c_l \cdot \tau_l$以上とれないため、最大性能は飽和する。

3.3 多重プログラミング

データを格納すれば、CUと独立に処理を実行するPLUの特徴を生かして、また逆に、データが出力されるまでの遅延時間を有効に使うために、多重プログラミングによる処理を考える。

図10に示すように、CUにおける処理時間をt、その確率分布をf(t)、PLUにおける処

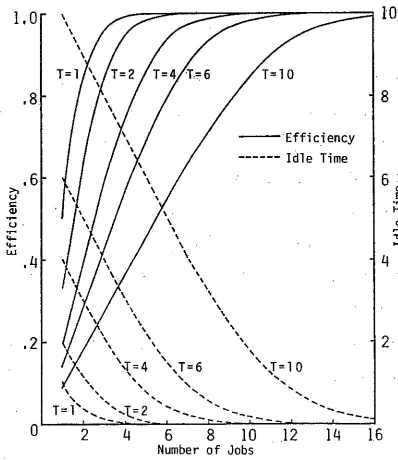

図11．ＣＵの効率と待ち時間

理時間をＴ、ジョブ切り換えのためのオーバヘッド時間をτとすると、ＣＵの待ち時間の平均T_iは、多重度をｎとして、

$$T_i=\int_0^{T-n\cdot\tau}(T-n\cdot\tau-t)\cdot f_{n-1}(t)dt \qquad (11)$$

で表される。ただし、Ｔ、ｆ（ｔ）はそれぞれ等しいとし、f_n（ｔ）はｆ（ｔ）のｎ回の畳み込みとする。いま、ｆ（ｔ）を平均1/λの指数分布と仮定すると、

$$f_n(t)=f(t)\circledast f(t)\circledast\cdots\circledast f(t)$$
$$=\lambda^n\cdot t^{n-1}\cdot\exp(-\lambda\cdot t)/(n-1)! \qquad (12)$$

で与えられる。ただし、⊛は畳み込み演算を表す。このときT_iは、

$$T_i=((n-1)/\lambda-a)\cdot e\cdot\left(\sum_{r=1}^{n-1}(\lambda^{r-1}\cdot a^{r-1}/(r-1)!)+1-e\right)$$
$$-e\cdot\lambda^{n-2}\cdot a^{n-1}/(n-2)! \qquad (13)$$

$$(a=T-n\cdot\tau,\ e=\exp(-\lambda\cdot(T-n\cdot\tau)))$$

となる。これを用いてＣＵの効率η、及びスループットTh_pを求めると、

$$\eta=1/(1+(n\cdot\tau+T_i)\cdot\lambda/n) \qquad (14)$$
$$Th_p=n/(1/\lambda+\tau+T_i/n) \qquad (15)$$

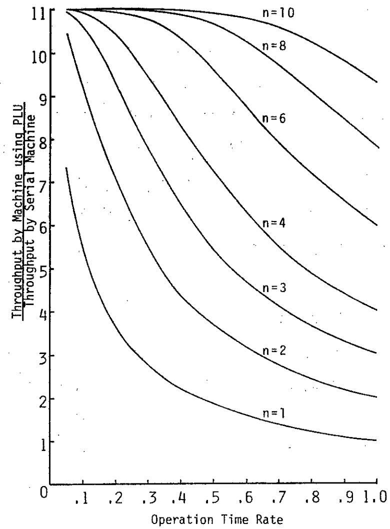

図12．ＰＬＵによる処理と逐次型処理の比較

となる。この様子を図11に示す。多重プログラミングによって、制御装置の空き時間を利用でき、一般の処理についても性能の向上が得られる。さらに、ＰＬＵの特徴により、各ジョブに割り当てられるＰＥ数は任意であり、汎用性に富んでいる。

比較対象として、逐次型処理による時分割ｎ多重を考える。１タスク当たりの制御に要する時間を1/μ、演算に要する時間を1/αとすると、スループットTh_sは、

$$Th_s=1/(1/\mu+1/\alpha+\tau) \qquad (16)$$

で表される。したがって、その性能比は、

$$\frac{Th_p}{Th_s}=\frac{1/\mu+1/\alpha+\tau}{n/\lambda+n\cdot\tau+T_i} \qquad (17)$$

となる。

1/λ＝1/μ、1/α＝ｋ・Ｔとしたときのスループットの比を図12に示す。ＰＬＵを用いることによるスピードアップｋと、多重化によるスピードアップとが相乗されて、有効に動作する。

以上の検討より、一般の処理プログラムの演算部分を高速化するためのハードウェア資源であるＰＬＵは、従来の計算機システムに組み込むことによって、処理全体の性能改善に役立つことを示した。その際、互いに独立に動作しているＰＬＵとＣＵの同期のためのオーバヘッドを解消するために、ＰＬＵの容量が十分大きければ、演算処理を多重化して、見かけ上、各装置が常に動作しているような方法を提案した。その結果多重度は、ＰＬＵ内にテンプレートを製作する時間とのトレード・オフ、及びＰＬＵとＣＵの実行時間の差によって、最適値を持つことを示した。オーバヘッドを小さくするために、繰り返し演算などによって、一つのテンプレートを数多く使用するようなスケジューリングが必要とされる。

４．おわりに

ノイマン型計算機において指摘されている問題点、とくにＣＰＵとメモリとの間のバス・ボトルネックを緩和し、高速演算を可能にするために提案されたＰＬＵについて、その構造、処理方式などを述べた。さらに、ＰＬＵを用いた計算機について、各レベルから並列性を考慮して、処理時間の検討を行った。ＰＬＵは、その内部は"垂れ流し"的データフローで動作するため、数式内に潜在する並列性を引き出すことも可能だが、外部との同期が問題となる。そこで、ＰＬＵを多重化して、ＣＵの待ち時間を減少し、システムのスループットを向上させることを考えた。簡単な解析によって、その効率、有効性などを検討し、演算に対して制御の割合が小さければ、ＰＬＵによる効果と相乗されることを示した。これは、装置のスピードアップだけではなく、条件文や繰り返し文の展開、ＰＬＵへのマッピングによっても達成され得る。

今後、シミュレーションなどによる詳細な解析を施すとともに、言語プロセッサの開発を行っていく予定である。

［参考文献］

[1] J.Backus："Can Programming Be Liberated from the von Neumann Style? A Functional Style and Its Algebra of Programs," CACM, vol.21.no.8, pp.613-641,Aug.1978.

[2] R.W.Hockney and C.R.Jesshope："Parallel Computers," Adam Hilger Ltd,Bristol,1981.

[3] 宇都宮公訓："データフロー計算機," bit誌,vol.12, No3-No.9(1983-02~08).

[4] 吉岡良雄："メモリステップにプログラム可能な演算機能をもつメモリ装置の提案," 昭58-10,信学論(D).

[5] 山田他："プログラム可能論理演算装置の構成と特性," 信学技報,EC83-25(10-1983).

[6] 山田他："プログラム可能論理演算装置," 第7回情報理論とその応用研究会資料,昭59-11.

[7] 青木他："プログラム可能論理演算装置の制御方法について," 信学技報,EC84-32(10-1984).

[8] Maley.G.A and Earle.J："The Logic Design of Transistor Digital Computers," Prantice-Holl, New Jersey,1963.