ロボットの動的制御における計算の並列化アルゴリズムとオブジェクト指向による実現*

田川 聖治**・神吉 良英***・太田 有三**・羽根田博正**

Parallel Processing Scheme for Dynamic Control of Robot Arm and Realization by Using Object Oriented Approach*

Kiyoharu TAGAWA**, Yoshihide KANKI***, Yuzo OHTA**
and Hiromasa HANEDA**

A parallel processing scheme is described for dynamic control computation of a robot-arm on any number of parallel processors. The control law for dynamic control of the robot-arm is generally represented by the sum of products. Such a control law usually consists of a huge number of operations. However, some of these operations are redundant and may be reduced by factorization. This paper proposes a practical scheduling algorithm which assigns the operations of the control law to parallel processors considering how many of them could be reduced by factorization.

Based on the proposed algorithm, a scheduling system is also developed by using an object oriented approach (g++). Then, a wonderful effect of the proposed algorithm is demonstrated through examples.

1. はじめに

ロボットアームの動的制御とは、ロボットアームの運動方程式（非線形モデル）に基づき制御系を設計することであり¹⁾、非線形特性による影響が無視できない高速、高精度な制御に有効である。ところが、動的制御の制御則は一般に長大であり、ロボットアームのオンライン制御を行うためには高速な演算処理が計算機に要求される。

本論文では、ロボットアームの制御則が多変数の積和式で与えられることに着目して、積和式の演算処理を複数のプロセッサに割り当てるためのアルゴリズムを提案する。演算式の並列処理については、計算木の高さの削減アルゴリズム（ツリーハイトリダクション）などがよく知られている^{2),3)}。ところが、2分木状の演算を効率よく行うためには、プロセッサ間の通信が自由に効率よく行える必要があるが、現実の並列計算機においては、プ

ロセッサ間の結合網はハードウェア量などの問題から制限されることが多い。さらに、ロボットアームの制御則の計算に必要な演算量は膨大であるため、従来の並列計算手法が有効に適用できるほど十分な数のプロセッサを確保することも現実的ではない。

このようにプロセッサの数や通信が制限されている場合、演算の並列性のみならず、Fig.1に示すような因数分解による冗長な演算の削除も考慮すべきである。特に積和式で与えられたロボットアームの制御則は、共通な

Fig. 1 Factorization

* 原稿受付 1994年1月20日

** 神戸大学 工学部 Faculty of Engineering, Kobe University; Rokkodaicho 1-1, Nada ward, Kobe city, Hyogo 657, JAPAN

*** 神戸大学 工学部 工学研究科 Graduate School of Engineering, Kobe University; ditto

Key Words : parallel processing, dynamic control of robot arm, factorization scheduling algorithm, object oriented.

因子を多く含んでいるので、因数分解による演算式の簡単化は重要である⁴⁾。

本論文で提案する積和式の並列化アルゴリズムは、与えられたロボットアームの制御則に含まれる共通因子を考慮してプロセッサに演算処理を割り当てるものであり、演算処理の並列化のみならず、因数分解による演算回数の削減も効果的に行うことができる。すなわち、各プロセッサによる逐次処理部分の計算効率を高めることにより、並列計算機における全体の処理時間を短縮するものである。

また、本論文では、提案したアルゴリズムに基づき、REDUCEなどの数式処理システムを用いて導出したロボットアームの制御則⁵⁾を自動的に並列化するシステムをオブジェクト指向により実現している。

以下、本論文の内容として、2.でロボットアームの制御則が積和式で表わされることを示し、3.で並列計算機モデルについて説明する。つぎに、4.で積和式の並列化問題を定式化すると共にアルゴリズムを提案する。5.では、オブジェクト指向による並列化システムの実現方法について述べる。さらに、6.では例題によって提案したアルゴリズムの有効性について検証する。最後に、本研究の成果をまとめると共に、今後の展開について触れる。

2. 動的制御におけるロボットの制御則

N 個の関節 (N 自由度) を持つロボットアームの運動方程式は次の (1) 式で表わされる。

$$f = M(q)\ddot{q} + C(\dot{q}, q)\dot{q} + D\dot{q} + g(q) \quad (1)$$

ただし、 $f \in R^N$ は関節に加える駆動トルクまたは力、 $q \in R^N$ は関節の変位を表わす。(1) 式右辺の第1項は慣性力を、第2項はコリオリ力と遠心力を、第3項は摩擦抵抗を、第4項は重力の影響を表わす。

80年代前半において、動的制御の主流は(2)式に示すような逆動力学計算による非線形フィードバックを行うことにより、ロボットアームを目標軌道と関節変位の関係において線形な動的システムに変換するものであった^{6)~8)}。

$$f = M(q)\ddot{u} + C(\dot{q}, q)\dot{q} + D\dot{q} + g(q) \quad (2)$$

ただし、 $\ddot{u} \in R^N$ は適当な制御量を表わす。

このため、逆動力学計算のための高速計算アルゴリズムの開発が盛んに行われ^{9), 10)}、複数のプロセッサによる並列処理に適した計算アルゴリズムも幾つか考案された^{11), 12)}。

逆動力学計算による動的制御の問題点は、ロボットアームの厳密なモデルが必要なことであった。そこで、80年代後半になると、ロボットアームの運動方程式に内在

する幾つかの重要な特徴に基づき¹³⁾、ロボットアームの運動方程式を決定するためのパラメータの値に誤差が含まれる場合においても、制御系の安定性がLyapunov関数により保証されるロバストな動的制御の研究が盛んに行われるようになった^{14)~16)}。これら新しい制御理論では、(2)式の逆動力学計算に代わり、(3)式に示す線形化補償器により非線形フィードバックを行っている。

$$f = M(q)\ddot{u} + C(\dot{q}, q)\dot{u} + D\dot{q} + g(q) \quad (3)$$

ただし、 $C(\cdot)$ は $\dot{M}(\cdot) - 2C(\cdot)$ が交代行列となるように選ぶものとする¹⁷⁾。

(3)式を用いた制御則の問題点は、 $C(\cdot)\dot{q}$ の代わりに $C(\cdot)\dot{u}$ を用いるためにNewton-Euler法¹⁰⁾など従来の逆動力学計算のための高速計算アルゴリズムが適用できないことである。

(2)式あるいは(3)式の制御則を数式処理システムによりスカラの演算式として導出すると、共に(4)式に示すような積和式の形で得られる。

$$f_n = \sum_{j \in J_n} T_j \quad (n = 1, 2, \dots, N), \quad (4)$$

$$T_j = \pm a_j \times \left(\prod_{m \in M_j} v_m^{h(m,j)} \right)$$

ただし、 a_j は定数パラメータ、 v_m は入力変数、 h は v_m の乗数である。 J_n は積項 T_j に対する添字集合の部分集合、 M_j は v_m に対する添字集合の部分集合である。

Fig. 2 Two-link robot arm

たとえば、Fig. 2 に示す2自由度のロボットアームの第2関節に対して(3)式の制御則を導出すると

$$f_2 = \sum_{j \in J_2} T_j, \quad J_2 = \{1, 2, \dots, 7\} \quad (5)$$

$$T_1 = b_3 \times \ddot{u}_1, \quad T_2 = L_1 \times b_4 \times c_2 \times \ddot{u}_1, \quad T_3 = b_3 \times \ddot{u}_2,$$

$$T_4 = L_1 \times b_4 \times s_2 \times \dot{q}_1 \times \dot{u}_1, \quad T_5 = D_2 \times \dot{q}_2,$$

$$T_6 = -G \times b_4 \times s_1 \times s_2, \quad T_7 = G \times b_4 \times c_1 \times c_2$$

となる。ただし、 c_n, s_n は三角関数 $\cos(q_n), \sin(q_n)$ を表わし、これらの値はあらかじめ求められているものとする。また、 b_m は基底パラメータ¹⁸⁾、 L_n は各リンクの長さ、 D_n は摩擦係数、 G は重力加速度を表わす。

3. 演算の並列処理方式

本論文では、(4) 式に示したロボットアームの制御則のうち、任意に与えられた関節 n ($1 \leq n \leq N$) に対して並列計算機による処理を考える。並列計算機は処理能力の等しい複数のプロセッサからなるものとし、さらに、各プロセッサは専用のメモリを持つものとする。並列処理は以下のように行う。まず、並列化するプロセッサの個数を P として、関節 n について (4) 式を次の (6)、(7) 式に示すように分割する。

$$\begin{aligned} f_{np} &= \sum_{j \in J_{np}} T_j, & (6) \\ p &= 1, 2, \dots, P, \\ J_n &= J_{n1} \cup \dots \cup J_{np} \cup \dots \cup J_{nP}, \\ J_{np} &\neq \phi, J_{np} \cap J_{nq} = \phi \quad (p \neq q) \end{aligned}$$

$$f_n = \sum_{p=1}^P f_{np} \quad (7)$$

並列計算機において P 個のプロセッサは、(6) 式に示した P 個の積和式の処理をそれぞれ行うものとする。一方、(7) 式の処理は、(6) 式の処理が終了した後、並列計算機におけるプロセッサの結合状態などを考慮して、適当なプロセッサによって行う¹⁹⁾。このとき、ロボットアームの制御則は一般に長大であり、プロセッサの個数 P に比べて (4) 式に含まれる積項の数が十分に多いため、(7) 式の処理時間とプロセッサ間の通信時間は相対的に無視できる。ただし、(4) 式の処理に必要なすべてのデータは、あらかじめ各プロセッサの専用メモリに貯えられているものとする。

4. 演算の並列化問題

本論文で考える演算の並列化問題は、与えられた関節 n に対して (4) 式の処理を (6) 式に示したように積項単位で P 個のプロセッサに割り当て、並列計算機における全体の処理時間を最短とすることである。

本論文では、プロセッサによる積項および積和式の処理時間を、それらに含まれる演算回数により評価するものとする。演算回数により処理時間を評価することは、従来の計算手法に関する研究と同じである²⁰⁻⁴⁾。また、乗算器を内蔵した最近のプロセッサにおいては、加減算と乗算の処理時間がほぼ等しいため²⁰⁾、演算回数による処理時間の評価は妥当であるといえる。

ここで、積項または積和式に含まれる定数および変数の総数を、それらのサイズ (size) と呼ぶことにする。ただし、 h ($h \geq 2$) 乗の変数が含まれる場合には、その変数を $(h-1)$ 回のかけ算に展開してサイズを評価する。このとき、積項あるいは積和式の演算回数はサイズの定

義より (サイズ-1) である。

4.1 因数分解を考慮しない並列化問題

因数分解による演算回数の削減を考慮しない場合、並列化問題は次のようになる。

【並列化問題 1】

入力：関節 n に対する (4) 式の積項 T_j ($j \in J_n$) とプロセッサの個数 P 。

目的：次式の値の最小値と、それを与える積項の P 個のプロセッサへの配分方法 ($J_{n1} \sim J_{nP}$)。

$$F_n = \max_{1 \leq p \leq P} \left\{ \text{size} \left(\sum_{j \in J_{np}} T_j \right) \right\}$$

ただし、

$$J_n = J_{n1} \cup \dots \cup J_{np} \cup \dots \cup J_{nP},$$

$$J_{np} \neq \phi, J_{np} \cap J_{nq} = \phi \quad (p \neq q)$$

である。□

並列化問題 1 は、タスク間の先行制約条件がないスケジューリング問題であり NP 困難である²¹⁾。この問題に対しては、精度のよい近似アルゴリズムとして次の LPT 法 (Longest Processing Time) がよく知られている²²⁾。

【並列化アルゴリズム 1】 (LPT 法)

手順 1 与えられた積項をサイズが大きな順に並べる。

手順 2 サイズの大きい積項から順に、その時点において次式の値が最も小さいプロセッサ p ($1 \leq p \leq P$) に割り当てる。

$$F_{np} = \text{size} \left(\sum_{j \in J_{np}} T_j \right) \quad \square$$

プロセッサの個数を $P=2$ として、(5) 式に示した 2 自由度のロボットアームの制御則に LPT 法を適用した結果を以下に示す。

$$\begin{aligned} f_{21} &= T_4 + T_7 + T_3 \\ &= L_1 \times b_4 \times s_2 \times \dot{q}_1 \times \dot{u}_1 + G \times b_4 \times c_1 \times c_2 + b_3 \times \ddot{u}_2 \end{aligned} \quad (8a)$$

$$\begin{aligned} f_{22} &= T_2 + T_6 + T_1 + T_5 \\ &= L_1 \times b_4 \times c_2 \times \ddot{u}_1 - G \times b_4 \times s_1 \times s_2 + b_3 \times \ddot{u}_1 \\ &\quad + D_2 \times \dot{q}_2 \end{aligned} \quad (8b)$$

(8a) 式には 10 回、(8b) 式には 11 回の演算が含まれている。各プロセッサに対して演算が均等に割り当てられていることから、(8) 式に示した積項の配分は、明らかに並列化問題 1 の最適解である。

4.2 因数分解を考慮した並列化問題

ロボットアームの制御則の逐次処理における計算効率を改善するため、我々は、積和式として与えられた制御則に対して繰り返し因数分解を行い、含まれる演算回数の総数が最少という意味で最適な演算式を自動的に生

成する最適化システムをすでに開発している⁴⁾。そこで、(8)式に示した二つの積和式を、この最適化システムにより因数分解した結果を以下に示す。

$$f_{21} = b_4 \times (L_1 \times s_2 \times \dot{q}_1 \times \dot{u}_1 + G \times c_1 \times c_2) + b_3 \times \ddot{u}_2 \quad (9a)$$

$$f_{22} = b_4 \times (L_1 \times c_2 \times \ddot{u}_1 - G \times s_1 \times s_2) + b_3 \times \ddot{u}_1 + D_2 \times \dot{q}_2 \quad (9b)$$

因数分解によって各々の積和式に含まれる演算回数は減少しており、最大の演算回数は(9b)式の10回である。しかし、この結果は必ずしも満足のいくものではない。たとえば、(5)式の積項を同じ二つのプロセッサに以下のように配分したとする。

$$\begin{aligned} f_{21} &= T_4 + T_3 + T_1 \\ &= L_1 \times b_4 \times s_2 \times \dot{q}_1 \times \dot{u}_1 + b_3 \times \ddot{u}_2 + b_3 \times \ddot{u}_1 + D_2 \times \dot{q}_2 \end{aligned} \quad (10a)$$

$$\begin{aligned} f_{22} &= T_6 + T_7 + T_2 \\ &= -G \times b_4 \times s_1 \times s_2 + G \times b_4 \times c_1 \times c_2 + L_1 \times b_4 \times c_2 \times \ddot{u}_1 \end{aligned} \quad (10b)$$

(10)式の積和式を同様に因数分解すると次の(11)式が得られ、最大の演算回数は9回となって(9)式の演算回数よりも少ない。

$$f_{21} = L_1 \times b_4 \times s_2 \times \dot{q}_1 \times \dot{u}_1 + b_3 \times (\ddot{u}_2 + \ddot{u}_1) + D_2 \times \dot{q}_2 \quad (11a)$$

$$f_{22} = b_4 \times (G \times (-s_1 \times s_2 + c_1 \times c_2) + L_1 \times c_2 \times \ddot{u}_1) \quad (11b)$$

そこで、与えられた積項をプロセッサに割り当てる際に、因数分解による演算回数の削減を考慮するものとして次の並列化問題を考える。ただし、因数分解後の積和式のサイズを $\text{size}(\text{fac}(\cdot))$ で表わす。

【並列化問題2】

入力：関節 n に対する(4)式の積項 T_j ($j \in J_n$) とプロセッサの個数 P 。

目的：次式の値の最小値と、それを与える積項の P 個のプロセッサへの配分方法 ($J_{n1} \sim J_{nP}$)。

$$E_n = \max_{1 \leq p \leq P} \left\{ \text{size} \left(\text{fac} \left(\sum_{j \in J_{np}} T_j \right) \right) \right\}$$

ただし、

$$\begin{aligned} J_n &= J_{n1} \cup \dots \cup J_{np} \cup \dots \cup J_{nP}, \\ J_{np} &\neq \phi, J_{np} \cap J_{nq} = \phi \quad (p \neq q) \end{aligned}$$

である。

□

並列化問題2において、すべての積項が互いに共通な因子を持たないとすれば、この問題を解くことにより並列化問題1の解が得られる。したがって、この並列化問題2もNP困難なクラスに属する極めて難しい最適化問

題であることがわかる。

4.3 因数分解を考慮したアルゴリズム

先に示したように、因数分解について考慮しない従来のLPT法では、並列化問題2に対して適切な解が得られない。そこで、LPT法を改良して並列化問題2に対する近似アルゴリズムを提案する。

提案するアルゴリズムでは、お互いに共通因子が少ない積項が異なるプロセッサに割り当てられるように、与えられた積項の共通因子を前処理によって調べる。この処理の基本的なアイデアを(5)式に示した制御則を例に説明する。(5)式から、節点を積項とし、辺を各積項間の共通な定数および変数とするグラフを作るとFig.3のようになる。このグラフの各節点(積項)に対し、以下の手順でラベル付けを行う。

Fig. 3 Terms and common factors in eq. (5)

【ラベル付けアルゴリズム】

- 手順1 ラベルが付いていない積項からサイズが最小な積項を選びラベル1とする。
- 手順2 ラベル1〜 k ($k \geq 1$)の積項と最大の共通因子を持つ積項のうち、サイズが最小な積項を選んでラベル $k+1$ とする操作を繰り返す。
- 手順3 手順2において共通因子を持つ積項が存在しない場合には、ラベル1〜 k の積項を思考の対象から除いて手順1に戻る。 □

Fig.3において、同じラベルを持つ積項の間には共通な定数や変数が少ないことがわかる。つまり、上に示したアルゴリズムに従い、すべての積項に対してラベル付けを行った後、同じラベルを持つ積項について集合を作ると、各集合には比較的に共通因子の少ない積項が集められることになる。そこで、各集合ごとにLPT法により積項をプロセッサへ割り当てると、同じ集合に含まれる積項は異なるプロセッサに分散され、共通因子が少ない積項が異なるプロセッサに割り当てられることが期待できる。

また、ラベル付けアルゴリズムにおいては、サイズの小さな積項を優先してラベル付けを行っているので、大きなラベルを持つ積項の集合には比較的サイズの大きな積項が集まる。したがって、LPT 法の効果を考慮し、大きなラベルを持つ積項の集合から順にプロセッサへの割り当てを考えればよいこともわかる。

さらに、提案するアルゴリズムでは、共通因子の多い積項ができるだけ同じプロセッサに配分されるように、因数分解を行った後の演算回数を予測して積項の割り当てを行う。すなわち、因数分解によって最適化した積和式に含まれる演算回数は、明らかに因数分解を行う前の積和式のサイズ ($\text{size}(\sum T_j)$) よりも小さく、また、積和式に含まれる異なる定数および変数の総数 ($\text{atom}(\sum T_j)$) よりも大きい。そこで、これらの平均値を取ることによって、積項を各プロセッサに割り当てる際に因数分解後の演算回数を推定する。

以上の考察をもとに、GCF/LPT (Greatest Common Factor/LPT) 法を以下に提案する。

【並列化アルゴリズム 2】 (GCF/LPT 法)

手順 1 与えられた積項にラベル付けを行う。

手順 2 同じラベルを持つ積項を一つの集合にまとめる。

手順 3 大きなラベルを持つ積項の集合から順に、各集合に対して以下の操作を行う。

手順 3.1 積項をサイズが大きな順に並べる。

手順 3.2 サイズの大きい積項から順に、その時点において次式の値が最も小さいプロセッサ p ($1 \leq p \leq P$) に割り当てる。

$$E_{np} = \frac{\text{size}\left(\sum_{j \in J_{np}} T_j + T_i\right) + \text{atom}\left(\sum_{j \in J_{np}} T_j + T_i\right)}{2}$$

□

ただし、 T_i ($i \notin J_{np}$) は、その時点で割り当てを考えている積項である。また、積和式に含まれる異なる定数および変数の総数を $\text{atom}(\cdot)$ により表す。

さきに、(10) 式で示した (5) 式の積項の配分方法は、GCF/LPT 法により得られたものである。

5. オブジェクト指向による実現

4. で提案したアルゴリズムに基づき、数式処理システムを用いて導出したロボットアームの制御則を、自動的に並列化するシステムをオブジェクト指向により実現した。システムの開発環境としては、コンパイラに g++ を、計算機に SPARC ワークステーションを使用した。このシステムでは積項やプロセッサをオブジェクトとして

Fig. 4 Structure of class

モデル化しているために、並列化アルゴリズムの試行錯誤的な改良が容易となっている。また、LPT 法を実現するためのクラスを継承して GCF/LPT 法のクラスを定義することにより、効率的なクラスの再利用も行っている。このように、オブジェクト指向を用いることにより柔軟性、拡張性の高いシステムが構築できた²³⁾。

Fig. 4 にクラス階層を示す。「symbol_table」は、積和式を構成する定数や変数を保持するデータ構造（多重リスト構造）を提供する基底クラスである。「equation」は並列化の対象となる積和式のオブジェクトを生成するクラスである。「term」は積項のオブジェクトを生成するクラスで、各積項のサイズとラベルをデータとして持っている。「LPT」と「GCF_LPT」は、それぞれ LPT 法と GCF/LPT 法の各操作をメソッドとするクラスである。「processor_F」と「processor_E」は、並列化する各プロセッサをオブジェクトとして生成するクラスであって、LPT 法と GCF/LPT 法における評価関数 F_{np} , E_{np} をメソッドとする。「term_list」は積項のプロセッサへの割り当て順序を記録するデータ構造を提供する。

6. 適用例によるアルゴリズムの検証

提案した GCF/LPT 法を、従来の LPT 法と比較することにより検証する。まず、Fig. 2 に示した 2 自由度のロボットアームの各関節の制御則に対して、GCF/LPT 法と LPT 法をそれぞれ適用して積項を与えられた個数のプロセッサへ配分した後、最適化システム⁴⁾により因数分解を行った。また、すべての積項の組み合わせについて因数分解を行うことにより最適解も求めた。これらの結果を Fig. 5 に示す。

Fig. 5 から、GCF/LPT 法によって得られた並列化

Fig. 5 Scheduling result of Fig. 2

Fig. 6 Three-link robot arm

Fig. 7 Scheduling result of Fig. 6

Table 1 Scheduling time

P	LPT (s)	GCF/LPT (s)
1	0.050	0.416
2	0.050	0.450
3	0.050	0.433
4	0.050	0.450

問題2の近似解は、あらゆる個数のプロセッサに対してLPT法を用いた場合よりも優れており、最適解に近接していることがわかる。

同様に、Fig. 6に示す3自由度のロボットアームのすべての関節に対して(3)式に示した制御則を導出し、GCF/LPT法とLPT法による並列化と最適化システム⁴⁾による因数分解を行った。Fig. 7に第2関節と第3関節についての結果を示す。また、Table 1は、第1関節についてGCF/LPT法とLPT法の処理時間を比較したものである。GCF/LPT法は、LPT法の約9倍の処理時間を要するが、十分に実用的なものであると言える。

7. おわりに

本論文で提案した並列計算手法ならびに並列化アルゴリズムは、積和式で表わされる任意の演算式に対して適用することができ、制御則の種類に依存しない²⁴⁾。また、対象とする並列計算機モデルが一般的であり、基本的には各プロセッサにおける逐次処理が中心となっているために、プロセッサの結合方法などハードウェアに大きく影響されることがない。したがって、トランスピュータなど多くの分散メモリ型並列計算機に対して有効であり、専用のVLSIプロセッサ²⁵⁾を用いる場合などに比べて、ほかの制御システムとの融合性や生産コストにおいて勝っていると言える。

今回、オブジェクト指向により開発したシステムは、数式処理システムの出力ファイルから積和式(制御則)を読み込み、並列化から因数分解までを自動的に行うことができるため、並列計算機のプログラム・コードを生成するような並列化コンパイラに発展させることも容易である。また、並列化アルゴリズムの改良を行う場合にも、クラスの再利用が可能である。

今後の課題としては、今回開発したシステムを用いてロボットアームに対する制御用プログラムを作成し、実際の並列計算機により評価することがある。

参 考 文 献

- 1) 古田：ロボットの動的制御；計測と制御，Vol. 30, No. 5, pp. 376~382 (1991)
- 2) D. J. Kuck：A Survey of Parallel Machine Organization and Programming；Computing Surveys, Vol. 9, No. 1, pp. 29~59 (1977)
- 3) 村岡：超並列処理コンパイラ，コロナ社，3章，pp. 28~64 (1990)
- 4) 田川，太田，藤原，羽根田：ロボットの逆動力学計算に対する最適化システム；計測自動制御学会論文集，Vol. 29, No. 2, pp. 220~226 (1993)
- 5) 斎藤，梅野，阿部，桑原，前川：REDUCEを用いたロボットアームの運動学および動力学方程式自動生成システム；電気学会論文誌C，Vol. 109, No. 2, pp. 89~96 (1989)
- 6) J. Y. S. Luh, M. W. Walker and R. P. C. Paul：Resolved-Acceleration Control of Mechanical Manipulators；IEEE Trans. on Automatic Control, Vol. AC-25, No. 3, pp. 468~474 (1980)
- 7) 古川：ロボットアームの位置と力の動的ハイブリッド制御；日本ロボット学会誌，Vol. 3, No. 6, pp. 531~537 (1985)
- 8) 小菅，古田，横山：ロボットの仮想内部モデル追従制御系；計測自動制御学会論文集，Vol. 24, No. 1, pp. 55~62 (1988)
- 9) J. M. Hollerbach：A Recursive Lagrangian Formulation of Manipulator Dynamics and Comparative Study of Dynamics Formulation Complexity；IEEE Trans. on System, Man, and Cybernetics, Vol. SMC-10, No. 11, pp. 730~736 (1980)
- 10) J. Y. S. Luh, M. W. Walker and R. P. C. Paul：On-Line Computational Scheme for Mechanical Manipulators；Trans. ASME J. of Dynamic Systems, Measurement, and Control, Vol. 102, pp. 69~76 (1980)
- 11) K. Hashimoto and H. Kimura：A New Parallel Algorithm for Inverse Dynamics；Int. J. of Robotics Research, Vol. 8, No. 1, pp. 63~76 (1989)
- 12) 橋本：逆ダイナミクスの新しい並列計算法；計測と制御，Vol. 30, No. 5, pp. 406~411 (1991)
- 13) 有本（システム制御情報学会（編））：ロボットの力学と制御，朝倉書店，1章，pp. 1~26 (1990)
- 14) D. Koditschek：Natural Motion for Robot Arms；Proc. of 23rd IEEE CDC, pp. 733~735 (1984)
- 15) J. J. E. Slotine and W. Li：On the Adaptive Control of Robot Manipulators；Int. J. of Robotics Research, Vol. 6, No. 3, pp. 49~59 (1987)
- 16) 吉川，井村，村井：ロボットマニピュレータのロバストな軌道追従制御；システム制御情報学会論文誌，Vol. 3, No. 7, pp. 218~225 (1990)
- 17) M. W. Spong and M. Vidyasagar：Robot Dynamics and Control, John Wiley & Sons, Theorem 6.3.1, pp. 141~143 (1986)
- 18) H. Mayeda, K. Yoshida and K. Osuka：Base Parameters of Manipulator Dynamic Models；IEEE Trans. on Robotics and Automation, Vol. RA-6, No. 3, pp. 312~321 (1990)
- 19) K. B. Irani and K. W. Chen：Minimization of Inter-processor Communication for Parallel Computation；IEEE Trans. on Computers, Vol. C-31, No. 11, pp. 1067~1075 (1982)
- 20) 井門，山田：トランスピュータ・ボードによる高速数値計算；インターフェース，Vol. 15, No. 5, pp. 251~258 (1989)
- 21) 野下，高岡，町田：基本算法，岩波書店，4章，pp. 255~257 (1985)
- 22) R. L. Graham：Bounds on Multiprocessing Timing Anomalies；SIAM J. Appl. Math., Vol. 17, No. 2, pp. 416~429 (1969)
- 23) 杉本：オブジェクト指向によるソフトウェア部品化と再利用支援環境；システムと制御，Vol. 31, No. 9, pp. 674~681 (1987)
- 24) K. Tagawa, Y. Kanki, Y. Ohta and H. Haneda：A Parallel Processing Scheme for Dynamic Control of Robotic Manipulators；Proc. of IECON '93, pp. 1419~1424 (1993)
- 25) Y. Fujioka, M. Kameyama and T. Higuchi：240MOPS Reconfigurable Parallel VLSI Processor for Robot Control；Proc. of IECON '92, pp. 1385~1390 (1992)