Vol. 31 No. 11 情報処理学会論文誌 Nov. 1990

共有メモリ型並列処理システムにおける変数の配置†

小池 稔†† 宮武克昌†† 大川善邦††

本論文では密結合型並列処理システムにおいて，プログラム上に現れる変数・配列変数等を複数の共有メモリに配置するアルゴリズムを提案する．共有メモリ上での変数の配置を考える場合，変数を1つの共有メモリへ集中すると，複数のプロセッサがその共有メモリをアクセスする時に，アクセスの衝突が起こる．逆に，変数を多くの共有メモリへ分散すると，交換網の切り替え回数が増大し，結果として処理時間が延びる．したがって，処理時間を最小にする最適の配置が存在するはずである．本論文で提案する配置アルゴリズムはクラスタリング法に属する1つのヒューリスティック解法である．まず，2つの変数が異なるタスクからアクセスされる可能性が高い時に値が大きくなり，同じタスクからアクセスされる可能性が高い時に値が小さくなるノルムを定義する．このノルムを使って，変数集合の中から1つの変数を選択し，これを順次共有メモリへ配置していくという手続きを取っている．さらに提案したアルゴリズムの妥当性を検証するために，ベンチマークテストを行う．例題として，歯車の設計問題を取り上げる．一方で，シミュレータを製作し，シミュレーション実験により，われわれが提案するアルゴリズムを使用した場合と，そうでない場合の定量的な比較を行う．共有メモリにおける変数の配置を考えることによって，全処理時間が14%も減少することが結果として得られた．

1. はじめに

ここでは，クロスバーあるいはこれと同等の機能を持つ結合網によって p 台のプロセッサと m 個の共有メモリが密結合されている並列処理システムを考える．最終的な目標はこのシステムの高水準言語コンパイラを作ることである．この目標を達成するために，多くの問題を解決していかなければならない．今回は，その中で特に，プログラム上に現れる変数・配列変数等を複数の共有メモリへどのように配置すればよいか，という問題を考える．

これまで，タスク（あるいはジョブ）をマルチプロセッサへ割り当てるためのスケジューリング方法は多く報告されている[1)~3)]．これらの方法を使えば，なんらかの基準に従って，与えられたタスクを複数のプロセッサへ割り当てることができる．しかし，並列処理システムを対象にしたコンパイラを構築する立場から考えると，タスクがプロセッサに割り当てられただけでは問題は解決されない．プログラム上に現れてくる変数をメモリに配置するアルゴリズムが必要である．単一プロセッサの場合であれば，変数のメモリ配置の方法を考える必要はない．たとえば，変数がプログラムに現れてくる順番に従ってメモリへ配置していけばよい．メインメモリが，ランダムアクセス機能を持つ以上，配置場所は結果に影響を与えない．

ところが並列処理システムになると状況は異なる．変数は物理的に異なる共有メモリへ置かれる可能性がある．たとえば，あるタスクが変数 A，B にアクセスするものとして，A と B が別の共有メモリに配置されていたとすると，タスクはまず A にアクセスし，その後に結合網を切り替えて B にアクセスしなければならない．逆に，2つのプロセッサからアクセスされるいくつかの変数が同一の共有メモリへ配置された場合は，メモリへのアクセスが同時に起これば，どちらか一方は待ち状態に入るという衝突が起こる．ここから，共有メモリへどのように変数を配置すればよいかという問題が発生する．これを，ここでは共有メモリにおける変数の配置問題と名付ける．

この問題は2つの相反する条件，(1)共有メモリへのアクセスの衝突回数を減少させるためには変数をできる限り多くの共有メモリへ広く分散させる方法がよい，と(2)結合網の切り替え回数を減少させるためには変数を1つのメモリへ集中しておいた方がよい，の間で平衡点を求めるスケジュール問題となる．

これまでに，Krause らはマルチプログラム環境でのスケジュール法[4)]を提案している．ここでは複数のタスクは始めから終わりまで唯一の共有メモリを占有するので，メモリの容量だけが制限条件になり，変数配置という問題はない．マルチプロセッサシステムの性能評価という面からのアプローチ[5),6)]もあるが，これは統計的な方法を密結合型並列処理システムへ適用し，その総合的な性能を決定するものであり，スケジ

† A Variable Allocation Algorithm in Shared Memory Parallel Processing Systems by MINORU KOIKE, KATSUMASA MIYATAKE and YOSHIKUNI OKAWA (Department of Mechanical Engineering for Computer-Controlled Machinery, Faculty of Engineering, Osaka University).

†† 大阪大学工学部電子制御機械工学科

ューリングという問題ではない．川口らは共有メモリ型マルチプロセッサにおいてジョブの総コストの下界値を求める方法[7]を提案している．これは，OS におけるスケジュール法の比較などに適用できる．

一方で，行列の要素をメモリの中に配置する方法に関する研究[8),9)]がある．これらは，配列変数を共有メモリの中に，どのように配置すればプロセッサがメモリにアクセスする時に衝突を避けることができるかという問題を扱っている．題目的にはここでの研究と共通点があるが，ここでは変数はベクトルのように整然とした配列とは考えていない．一般のタスクを考え，それらが任意の時点で，任意の変数にアクセスする状況を考える．

さらに，ディジタルシステムの論理設計の自動化の際のデータパス合成に関する研究[10),11)]がある．これらは，制御/データフロー表現からデータパス構造への変換において，基本動作系列に現れる変数をできるだけ少ない数のレジスタへ割り付けるという問題を扱っている．レジスタをメモリと見なせばここでの研究と共通点があるが，これらの問題が必要なレジスタ数の最小化を目的としているのに対して，ここではアルゴリズムに対する初期条件としてメモリ数が与えられ，各変数をこの値の数のメモリに必ず配置することを目的としている点が異なる．

これに対して，われわれがここで提案する変数の配置問題は，既に述べたように，コンパイラ指向のスケジュール問題であり，メモリアクセスの衝突と結合網の切り替えという2つの条件を考慮して，変数配置の具体的な案を導出するものである．

2. 問題の定義

p 台のプロセッサと m 個のメモリが，**図1**に示すように，クロスバー結合網によって結合されているシステムを考える．1つの計算問題は n 個のタスクから構成され，その先行制約関係はタスクグラフ（図3はタスクグラフの一例）という形であらかじめ与えられ，かつ各タスクの処理時間は明示的に与えられていると仮定する．

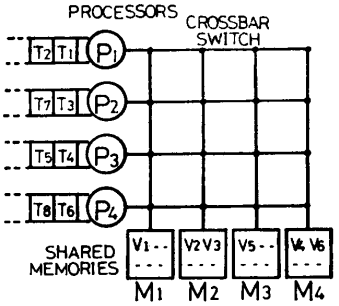


図 1 共有メモリ型並列処理システムの構成
Fig. 1 A configuration of shared memory parallel processing systems.

この条件のもとで，各タスクをプロセッサへ割り当てる問題は静的なタスク・スケジューリング（static task scheduling）問題といわれている．本論文で取り扱う問題は，静的なスケジューリング問題に対して，さらに共有メモリの条件を組み込むので，これを一般的に解くとなると問題の規模が大きくなり，結果を得るまでの計算時間が非現実的に増大する可能性がある．これを避けるために，ここでは，分割統治の原則（divide and conquer）を採用する．すなわち，タスクをプロセッサへ割り当てる問題と変数を共有メモリへ配置する問題を連立した形で解くのではなく，まず第1段階において，変数のメモリ配置を無視してタスクの割り当てを決め，第2段階において，タスク割り当てを与えられたものとして，それに最も適するメモリ配置を考える，という順序で問題を前後に分離した形で解を求める．

以下で，本論文で用いる記号の説明を行う．1つの計算問題は n 個のタスク T_k $(k=1,2,\cdots,n)$ から構成される．タスク T_k の処理時間は Q_k である．Q_k は共有メモリにおける衝突による待ち時間および結合網の切り替え時間を含まない純粋な処理時間である．タスクの先行制約関係はタスクグラフとして与えられる．これまでに提案されているアルゴリズムの1つを使って，n 個のタスクを p 台のプロセッサ P_h $(h=1,2,\cdots,p)$ へ割り当てる．添字の集合を a_h とする時に，$k\in a_h$ なる時 T_k は P_h に割り当てられたという．以下で，a_h は既知とする．

一般に a_h が与えられただけでは，タスクの開始時間は定まらない．クリティカルパス上のタスク以外は前後に余裕時間を持っている．ここでは，タスクは実行のための条件が整えば直ちに実行される（前倒し方式）とする．タスク T_k の実行を開始する時刻を t_{ks}，終了時刻を t_{kf} とする．また，T_1 は先頭のタスク，T_n は最後のタスクとなるようにラベルを付ける．全処理時間は

$$t_{\text{total}}=t_{nf}-t_{1s}$$

と表される．

ここで考える並列処理システムにおける処理は同期して実行されるとする．時間はそれ以上分割すること

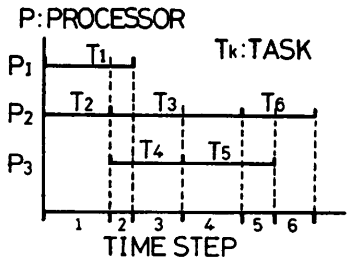


図 2 タスクの実行とタイムステップ
Fig. 2 Execution of tasks and the setting of time step.

ができない基本的な単位時間（基本時間）を持つ．メモリアクセス等の処理要求は基本時間の最初において発生すると仮定する．タスク T_k の処理時間 Q_k は基本時間の整数倍として与えられる．

$2n$ 個の t_{ks} と t_{kf} を単調非減少の順序に並べる．たとえば

$$t_{1s}=t_{2s}<\cdots<t_{nf}$$

のようになる．ここで等号が成り立つ時刻は1つの記号を代表として残し，その他は除外する．この時刻の集合をイベント集合と呼ぶ．イベント集合に属する時刻は少なくとも1つのタスクが開始されるか，あるいは終了するかの事象が発生している．イベント集合によって互いに疎に切られる時間帯をタイムステップと呼ぶ．タイムステップの一例を**図2**に示した．

プログラム上に現れる v 個の変数を V_i $(i=1,2,\cdots,v)$ とする．ここで，$v>m\geqq 2$ とする．$m=1$ ならば，メモリ1個で配置を考える必要がない．変数の数はメモリの数よりも十分に大きいと仮定する．もし $v\leqq m$ ならば，各変数は別々の共有メモリへ配置すればよい．タスク T_k がタイムステップ s において，変数 V_i にアクセスする回数を c_{iks} とする．以下で，c_{iks} は既知とする（たとえば，プログラムのループの解析等の方法[12]により）．添字の集合 b_g $(g=1,2,\cdots,m)$ がありかつ $i\in b_g$ の時，V_i はメモリ M_g に配置されたという．ここで考える変数の配置問題は

$$\min_{b_g}\ (t_{nf}-t_{1s})$$

なる b_g を求める問題と定式化できる．ここで，基本時間の最初において，複数のタスクが同一の共有メモリへアクセスする要求を出した時は，1つの要求のみが満足され，それ以外の要求は次の基本時間の開始点まで待たされる．あるタスクが1つの共有メモリへアクセスし次に別のメモリへアクセスする時は，結合網のスイッチの切り替えのための時間だけメモリアクセス時間が長くなる．これが最適解を求めるための制約条件である．

3. 変数のメモリ配置アルゴリズム

2つの変数 V_i，V_j 間の距離（ノルム）d_{ij} を次式によって定義する．

$$d_{ij}=\sum_{s=1}^{q}\sum_{\substack{k=1,\\ c_{iks}\neq 0}}^{n}\sum_{\substack{h=1,\\ h\neq k,\\ c_{jhs}\neq 0}}^{n}(c_{iks}+c_{jhs}) \qquad (1)$$

2つの変数 V_i と V_j が異なるタスクから同じタイムステップにおいてアクセスされる可能性が高い時，ノルム d_{ij} は大きくなり，同一のタスクからアクセスされる可能性が高い時には d_{ij} は小さくなる傾向を示す．ただし，q はタイムステップの総数である．

d_{ij} を既知として，以下のアルゴリズムにより v 個の変数を m 個のメモリへ配置する．

変数の共有メモリへの配置アルゴリズム

（0） 初期化

$B=(1,2,\cdots,v)$：変数の添え字の集合

$b_g=\phi$（空集合）：b_g は g 番目の共有メモリへ割り当てられた変数の添え字の集合

（1） 最初の配置

すべての共有メモリへ各々1個の変数を配置する．

① $\max_{i,j} d_{ij}$ となる i，j を見つける

$i\rightarrow b_1$，$j\rightarrow b_2$，$B-(i,j)\rightarrow B$，$3\rightarrow g$ とする

② $g\leqq m$ ならば ③，そうでなければ（2）

③ 選択：$\max_{i\in B}\sum_{j\in B} d_{ij}$

変更：$i\rightarrow b_g$，$B-i\rightarrow B$，$g+1\rightarrow g$

goto ②

（2） 判定

B が空集合の時，計算終わり，そうでなければ（3）

（3） 配置

選択：$\max_{i\in B}\max_{g}\sum_{j\in b_g} d_{ij}$ なる変数 i を選択

配置：その変数 i を $\min_{g}\sum_{j\in b_g} d_{ij}$ なるメモリ M_g に配置

変更：$B-i\rightarrow B$，$b_g+i\rightarrow b_g$

goto（2）

このアルゴリズムは本質的にクラスタリング法[13]に属し，変数間距離 d_{ij} により変数を整列させ，その順序で変数を配置していくというヒューリスティック解法[14]である．

4. シミュレータの製作

ここで提案したアルゴリズムの妥当性を検証するために，密結合型の並列処理システムのシミュレータを製作した．以下に，その概要を箇条書きにする．

A. 前提条件

(1) 各タスクの計算手続きはアセンブラ・プログラムで与えられる．プログラムはプロセッサのローカル・メモリへ格納される．データは変数として共有メモリに格納される．つまり命令とデータは物理的に分離されている．

(2) タスクのプロセッサへの割り当ておよび変数の共有メモリへの配置は既に決まっている．

(3) タスクを開始する条件は与えられている（前倒し方式）．

(4) プロセッサは1つのタスクを開始した時，それが終了するまで別タスクを開始することはない（イベント駆動）．

(5) プロセッサが他のプロセッサのタスクの終了を待っている場合，もしマルチジョブがサポートされているならば，そのプロセッサはこのジョブ以外の他のジョブのタスクを実行するはずである．他のジョブのタスクは，機械語レベルで考えた場合には様々な命令を使っていて，それを克明にシミュレートすることは不可能である．そこでここでは，命令列を統計的に平均化した状態で扱うために，タスクの終了待ちになったプロセッサは一定のアイドル・タスクを実行すると仮定している．このアイドル・タスクは $(1+F)$ クロックの周期的な命令を繰り返すもので，1クロックの共有メモリへのアクセスと，F クロックの CPU の内部（共有メモリにアクセスしない）仕事からなる．

B. シミュレータの機構

(1) システム全体は基本的なクロックを有し，これに同期してすべての動作が進行する．

(2) 命令実行のメカニズム

1つの命令のクロック数は次式によって与えられる．

$$u=A\cdot x+B\cdot y+C+D\cdot z \qquad (2)$$

ここで，A, B, C, D は使用する CPU によって決まる定数である．A は転送，B は演算，D は初等関数演算（sin など）を表す係数で，C は固定クロック数である．また，x, y, z は使用する命令ごとに変化する数値である．たとえば，8086 と 8087 を組み合わせたシステムをシミュレートする場合は

$A=30$, $B=50$, $C=15$, $D=150$

とし，命令が浮動小数点演算で FADD mem の場合は

$x=1$, $y=1$, $z=0$

となり，全体で $u=95$ クロック必要ということになる．以下同様である．

(3) 共有メモリ・アクセスのメカニズム

プロセッサがアクセスしようとした共有メモリが既に他のプロセッサによって占有されている時は，そのプロセッサは共有メモリが開放されるまでそのメモリに用意された待ち行列へ入る．

(4) 結合網切り替えのメカニズム

結合網の状態を変化させる時にはスイッチの切り替えのための時間を必要とする．

C. 計測値

(1) 全処理時間 t_{total}

(2) 共有メモリにおける衝突の回数 N_{col}

(3) 共有メモリにおける衝突によるプロセッサの待ち時間 t_{wait}

(4) 結合網の切り替えの回数 N_{sw}

5. 実験結果

3章で提案したアルゴリズムの妥当性を検証するためにシミュレーション実験を行った．ベンチマーク・テストとして機械工学において使用されている歯車の設計問題を取り上げた．計算式等は省略するが，この問題は浮動小数点演算を主とする70個のタスク，および137個の変数から構成されている．タスク間の先行制約を与えるタスクグラフは**図3**に示した．

ここで提案するアルゴリズムの有効性を実証するために，変数を乱数に従って共有メモリへ配置した場合(以下でランダム配置と呼ぶ)と，3章のアルゴリズムを使って配置した場合（以下で最適配置と呼ぶ）の両者の結果を比較した．ここで，転送・演算比は 10，アイドル・タスクの $F=40$（1の共有メモリアクセス，40 の内部処理）と固定した．**図4**(a)はランダム配置の場合の全処理時間，(b)は最適配置の場合の全処理時間の変化を示している．いずれの場合も，横軸は使用した共有メモリの個数，縦軸はプロセッサの台数を示している．全処理時間は等高線として与えられている．両図において，プロセッサの台数および共有メモリの個数が増加するに従って，全処理時間は短くなる傾向を示している．また，等高線の上に書かれた数値を比較することによって，最適配置を行った時の全処理時間はランダム配置のそれよりも短くなる傾

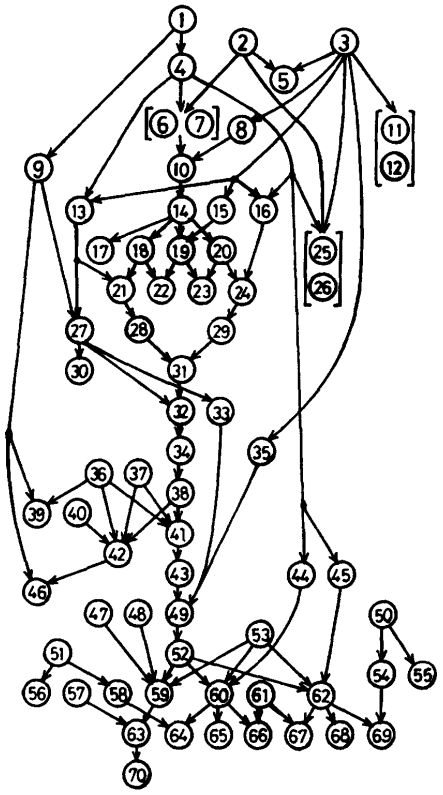


図 3 タスクグラフの例
Fig. 3 An example of the task graph.

向を示していることが分かる．

最適配置とランダム配置の性能を定量的に測定するために，最適配置の値をランダム配置の場合のデータで割り，100分比で示すことにした．すなわち，全処理時間の場合は

$$\tau_{total}=\frac{t_{total}(\text{最適配置})}{t_{total}(\text{ランダム配置})}*100(\%) \qquad (3)$$

となる．これを全処理時間比と呼ぶ．切り替え回数比 n_{sw}，衝突回数比 n_{col}，待ち時間比 τ_{wait} についても同様である．

図4(a),(b)に対して，(3)式を適用した結果を**図5**に示す．この図からみると，プロセッサと共有メモリの台数が等しい時に良い性能が得られていることが分かる．(3,3) の時に τ_{total} は 90.09%，(4,4) の時に τ_{total} は 89.61%，(6,6) の時に τ_{total} は 86.36% となっている．このバランスを崩して，たとえば，プロセッサを3台，共有メモリを2個とすると τ_{total} は 99% となり，メモリ配置の効果がないことを示している．プロセッサ2台メモリ8個の場合も 97.5% と配置の効果が見られない．

メモリの個数が少なければ，どのように配置しても

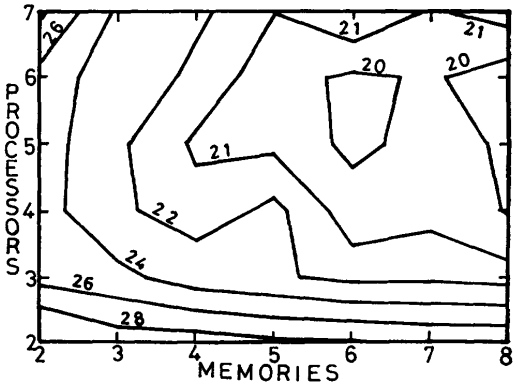


図 4(a) ランダム配置の全処理時間
(単位は 10^3 クロック)
Fig. 4(a) A distribution of the total processing time for a random variable allocation (Unit 10^3 clock).

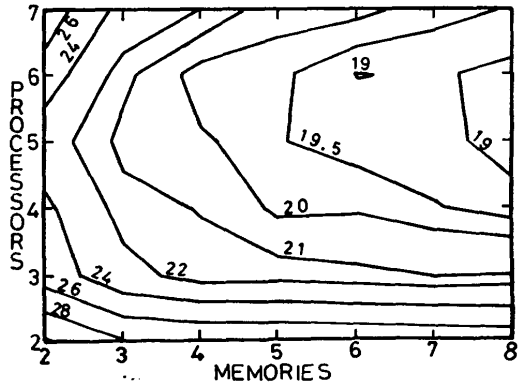


図 4(b) 最適配置の全処理時間
(単位は 10^3 クロック)
Fig. 4(b) A distribution of the total processing time for an optimized variable allocation (Unit 10^3 clock).

結局メモリのアクセス競合は起こる．逆に，メモリが多過ぎればアクセス競合は起こらない．これらの場合にはランダム配置と最適配置の差は少ない．しかし，並列処理の常識から言うと，プロセッサの台数とメモリの個数をアンバランスにすることはほとんどない．それから言えば，ここで提案したアルゴリズムは実用的な局面の多くをカバーしていると言える．

図6は同じ実験におけるメモリアクセス時の衝突の回数を示したものである．プロセッサ台数が減少または共有メモリ個数が増加するに従って衝突回数が少なくなる傾向を示している．**図7**は衝突の発生によるプロセッサの待ち時間を積算したデータである．(6,6) の場合には待ち時間は 45% も減少している．**図8**は結合網の切り替えの回数の比を示したものである．

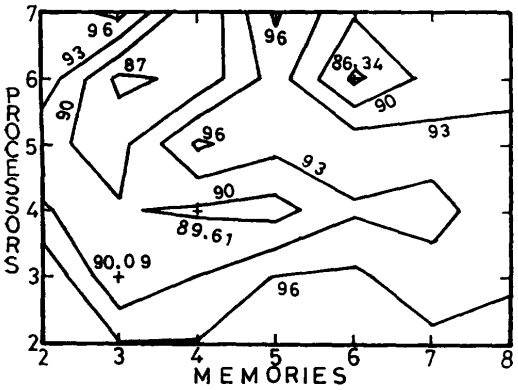


図 5 全処理時間比 τ_{total} (単位は %)
Fig. 5 A distribution of the total processing time ratio τ_{total} (Unit %).

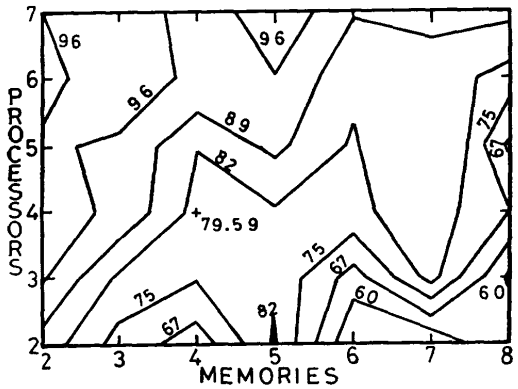


図 6 衝突回数比 n_{col} (単位は %)
Fig. 6 A distribution of the collision ratio n_{col} (Unit %).

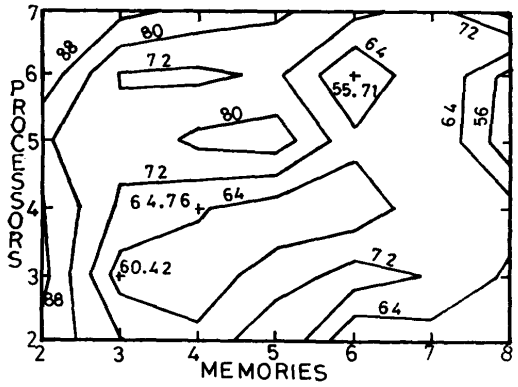


図 7 待ち時間比 τ_{wait} (単位は %)
Fig. 7 A distribution of the wait time ratio τ_{wait} (Unit %).

CPU 環境は(2)式によって与えられるが，A/B は共有メモリからデータをフェッチする時間（結合網における遅れ時間を含む）と，CPU 内で演算処理をする時間の比であって，使用する CPU とメモリの性能

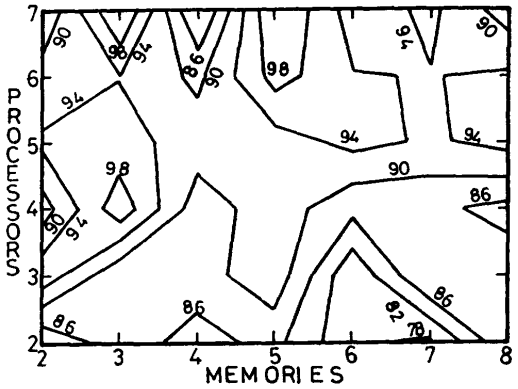


図 8 切り替え回数比 n_{sw} (単位は %)
Fig. 8 A distribution of the switching ratio n_{sw} (Unit %).

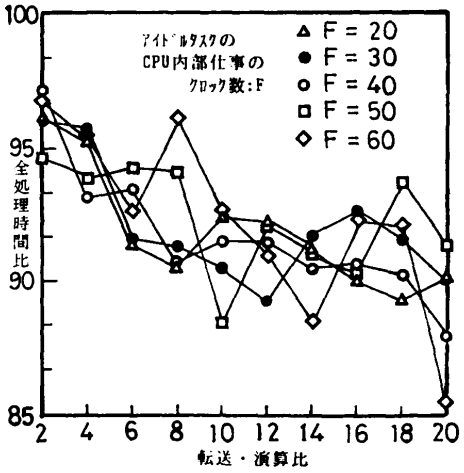


図 9 転送・演算比の全処理時間比に対する影響 (単位は %)
Fig. 9 Variation of the total processing time ratio according to the transfer-arithmetic ratio (Unit %).

を表す代表的なパラメータである．これを転送・演算比と呼ぶ．LSI 技術の進歩を予測すると，メモリからの転送時間に対し浮動小数点演算時間の高速化の方が早く進む可能性がある．A/B は現存の CPU に関する値だけではなく，将来の CPU 性能なども考慮して変化させる．

図 9 は転送・演算比を変化させて，全処理時間比 τ_{total} がどのように変化するかを示したものである．なお 5 本の折れ線はアイドル・タスクのパラメータ F を変化させたものに対応している．図において，転送・演算比が増加すると（横軸で右側へ進むに従って），τ_{total} が減少することが示されている．転送・演算比が大きくなるということは，メモリへのアクセス

時間が相対的に大きくなるということを意味するので，共有メモリへのアクセスの頻度が増す．その結果，ここで提案したアルゴリズムの有効性がより鮮明になることを示している．

逆に，メモリへのアクセスに比較して演算時間が長い CPU を使う場合（たとえば浮動小数点演算をソフトで組むような場合），あるいは共有メモリへのアクセス頻度が低い問題を計算する場合などは，変数を共有メモリに配置する問題はあまり重要ではない．単一プロセッサの場合のように変数が出てきた順にメモリへ配置するのがよいということになる．ただし，ここではプログラムはローカルメモリに格納されるという前提を置いていることに注意する必要がある．もし，プログラムが共有メモリ内に置かれるとすると，共有メモリのアクセスの回数は飛躍的に大きくなるので，その場合はメモリ配置問題が重要になる．

現在市販されている CPU 80386 とコプロセッサ 80387 の組合せを考えると，転送・演算比はおよそ 6（図 9 の横軸左からほぼ 1/3 のところ）となる．これから考えると，われわれが提案したアルゴリズムを使うと，変数をランダム配置した場合に比較して，7％程度処理時間が短縮されることになる．

6. おわりに

1つの計算問題を密結合型の並列処理システムを使って解く場合，プログラム上に現れてくる変数（実質的には配列変数）を共有メモリ上にどのように配置したらよいかという問題を考えた．この問題は単一プロセッサの場合には表面に現れてこなかった問題である．共有メモリが複数個あるために，これらのメモリのどれにどの変数を配置するかが処理時間に影響を与える．変数の共有メモリへの集中の度合をパラメータとして，トレードオフの関係にあるアクセスの衝突と結合網の切り替え回数を，共に適度に減少させる最適の配置が存在するはずである．これが，ここで提起した変数の配置の問題である．

この配置の問題はタスクのプロセッサへの割り当て問題と，変数のメモリへの配置問題という 2 つの問題を含んでいる．前者に対しては従来法を用い，ここでは後者の変数を共有メモリ上へ配置するアルゴリズムを提案した．まず，2 つの変数間にノルムを定義する．このノルムを使って変数集合の中から 1 つの変数を選択し，これを順次共有メモリへ配置していくという手続きを取っている．

アルゴリズムの妥当性を検証するために，ここではベンチマーク・テスト法を採用した．歯車の設計問題を選び，これを中粒度のタスクグラフとして表した．各タスクの中の計算手続きをアセンブラプログラムで与え，シミュレータに掛けて全処理時間等のデータを採取した．共有メモリ上における変数の配置を工夫することによって，全処理時間が 14% 程度も短縮されることが明らかになった．

ここで提案したアルゴリズムはタスクのプロセッサへの割り当てが決まった時に，変数を共有メモリ上へ配置するのに有効なアルゴリズムである．もし，共有メモリの数を変えて，その都度変数配置を行いシミュレーションを繰り返し，それらの結果を比較することができれば，最適の共有メモリの数とその配置が同時に得られることになる．

このアルゴリズムは計算手続きがあらかじめ明確に分かっている問題に対して並列処理の処理時間を短縮するのに有効なアルゴリズムと考えられる．今後の問題としては，複数のバスとキャッシュを持った並列処理システムに対して，適用することを考えている．

参考文献

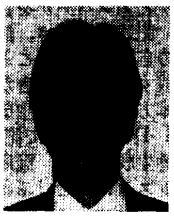
1) Ramamoorthy, C. V., Chandy, K. M. and Gonzalez, Jr., M. J.: Optimal Scheduling Strategies in a Multiprocessor System, *IEEE Trans. Comput.*, Vol. C-21, No. 2, pp. 137-146 (1972).
2) 笠原博徳，成田誠之助：マルチプロセッサ・スケジューリング問題に対する実用的な最適及び近似アルゴリズム，信学論 (D)，Vol. J 67-D, No. 7, pp. 792-799 (1984).
3) Lo, V. M.: Heuristic Algorithms for Task Assignment in Distributed Systems, *IEEE Trans. Comput.*, Vol. 37, No. 11, pp. 1384-1397 (1988).
4) Krause, K. L., Shen, V. Y. and Schwetman, H. D.: Analysis of Several Task-Scheduling Algorithms for a Model of Multiprogramming Computer Systems, *J. ACM*, Vol. 22, No. 4, pp. 522-550 (1975).
5) 布谷嘉章，住田修一，橋田 温：マルチプロセッサシステムの性能評価，信学誌，Vol. 66, No. 12, pp. 1261-1266 (1983).
6) 池原 悟：マルチプロセッサ方式における共用メモリアクセス競合の解析，信学論 (D)，Vol. J 63-D, No. 4, pp. 334-341 (1980).
7) 川口 剛，谷口祐治，喜屋武盛基：共有メモリ型マルチプロセッサにおけるスケジューリング，信学論 (D)，Vol. J 71-D, No. 11, pp. 2250-2258 (1988).
8) Lawrie, D. H.: Access and Alignment of Data

in an Array Processor, *IEEE Trans. Comput.*, Vol. C-24, No. 12, pp. 1920-1928 (1975).
9) Balakrishnan, M., Jain, R. and Raghavendra, C. S.: On Array Storage for Conflict-Free Memory Access for Parallel Processors, *Proc. 1988 Int. Conf. on Parallel Processing*, Vol. 1, pp. 103-107 (1988).
10) Tseng, C.-J. and Siewiorek, D. P.: Facet: A Procedure for the Automated Synthesis of Digital Systems, *20th Design Automation Conf.*, pp. 490-496 (1983).
11) 田中敏明, 小林 勉: FORTRAN 記述からのデータパス合成法, 信学論 (A), Vol. J71-A, No. 4, pp. 973-981 (1988).
12) Polychronopoulos, C. D.: *Parallel Programming and Compilers*, Kluwer Academic Publishers, Boston (1988).
13) Williams, E. A.: Design, Analysis, and Implementation of Distributed Systems from a Performance Perspective, Ph. D. Dissertation, Univ. Texas, Austin (1983).
14) 大附辰夫：大規模組合せ問題におけるヒューリスティック算法, 信学誌, Vol. 58, No. 4, pp. 416-423 (1975).

(平成2年1月18日受付)
(平成2年7月10日採録)

小池 稔 (正会員)

1963年生. 1986年大阪大学工学部機械工学科卒業. 1988年同大学院工学研究科産業機械工学専攻修士課程修了. 現在同大学院博士課程に在学中. 並列計算機におけるソフトウェア開発環境の研究に従事. 日本機械学会, 日本ロボット学会各会員.



宮武 克昌

1967年生. 1990年大阪大学工学部電子制御機械工学科卒業. 現在同大学院工学研究科電子制御機械工学専攻修士課程に在学中. 宇宙構造物の運動解析および軌道制御, 姿勢制御に関する研究に従事.

大川 善邦 (正会員)

1934年生. 1957年東京大学工学部機械工学科卒業. 1962年東京大学大学院数物系研究科博士課程修了. 工学博士. 1985年より大阪大学工学部電子制御機械工学科教授. 並列処理, 分散オペレーティングシステム, リアルタイム制御システムなどに興味をもっている. 計測自動制御学会, 人工知能学会, ACM, IEEE Computer Society などの会員.