段取り時間を考慮した細粒度タスクスケジューラ

正員 近藤 仁志 (東工大)
非会員 細井 隆史 (東工大)
非会員 原 辰次 (東工大)

Finely-Grained Task Scheduler Considering the Preparation Time

Hitoshi Kondoh, Member, Takafumi Hosoi, Non-member, Shinji Hara, Non-member (Tokyo Institute of Technology).

In parallel processing, it is more efficient to handle a job by decomposing it into finely-grained tasks than trying to manipulate the larger ones. However, if the tasks are finely grained, the overhead related to the synchronization of the processors and the shared memory accesses correspondently increases.

We propose an approach for reducing this overhead by constructing a parallel scheduler which considers the preparation time during plural searchings. In this scheduler, (1) the processes share two upper bounds derived by each searching, (2) the lower bound function is built taking into account the preparation time, and (3) the branching in the search tree is made by a heuristic node selection considering the node level.

An application of the proposed scheduler based on a multi-DSP system to the computation of the inverse dynamics for robot arms shows that the scheduling time can reduced to be more than twice shorter than that of the conventional scheduler, for the case of four processors.

キーワード：並列処理，分枝限定法，NP 困難，マルチプロセッサ，タスクスケジューリング

1. はじめに

マルチプロセッサ方式の性能を左右するスケジューリング問題は NP 完全⁽¹⁾であり，タスク数の多項式オーダーでの最適解の導出は保証されない。これに対し，実用的な近似最適解の導出手法はこれまで，ヒューリスティック法をはじめシミュレーテッド・アニーリング法⁽²⁾や遺伝的アルゴリズム⁽³⁾など様々なアプローチが行われ，近年数%といったスケジューリング精度を競うところまできている。そこで最近では，スケジューリングの精度だけでなく，タスクグラフのクリチカルパス長を最小化する方法など，プログラムの分割方法についても検討されるようになってきた⁽⁴⁾。

並列処理効率の面からいえば，プログラムは細かく分割したほうが効率的である。しかし，近年標準的に搭載されているパイプライン方式のプロセッサにおいては，共有メモリのアクセス時間とプロセッサ内のパイプ処理時間に著しい差があるため，長い式を分割するとオーバーヘッドが増える問題があった。

そこでここでは，タスクを一度演算レベルまで細かく分割し，これをスケジューリングの際に前後のタスクの組合せを考慮して，オーバーヘッドの最小化を図るスケジューラに

ついて検討する。ここでは，通信に共有メモリーを用いた場合と，同一プロセッサ上でレジスタを用いた場合との処理時間の差を段取り時間として表し，これを含めた処理時間全体を最小化する。

しかしながら，段取り時間の考慮によって，スケジューリング問題の計算複雑度は著しく悪化する。本論文ではまず，段取り時間を考慮したスケジューリングの問題規模が，組合せのオーダーから順列オーダーへ悪化することを示す。次に，このような問題でも効率的なスケジューリングが行えるようにするために，大規模問題に適しているとされる「複数の探索則とシャフリングを併用した並列型分枝限定アルゴリズム」⁽¹¹⁾を適用する。本論文ではこの手法を適用するにあたり，分枝限定法の限定操作を促進する二つの方法を提案する。一つは下限関数に段取り時間を考慮する方法。そして二つめは，ノードセレクションにおけるヒューリスティック値を優先する探索順序の決定法である。これを実際にマニピュレータの逆動力学計算⁽⁵⁾⁻⁽⁷⁾に適用し，細粒化によるクリチカルパス長の短縮化を図り，その効果を確認する。

以下，第2章では細粒度並列処理の問題点について述べ，第3章で段取り時間を考慮した場合のスケジューリング問題の定式化を行うとともに，問題の計算複雑度を考察

する。そして第4章では、この問題を解くためのスケジューリング法を提案し、第5章でその効果をみる。ここでは、乱数で発生した疑似タスクグラフへの適用結果と3軸ロボットアームの逆動力学計算の並列処理という実問題によって、その有効性を確認している。

2. 細粒度並列処理の問題点

〈2・1〉 実現上の問題点 プログラムを細かく分割してゆくと、タスク数の増加によりスケジューリング問題の計算複雑度が悪化することに加えて、並列処理することによって生じるオーバーヘッドが無視できなくなる。特に細粒度では、スケジューリングの実行精度が重視されるため、一般にバス密結合型並列処理システムで行われることが多い。このようなシステムでは、共有メモリー・アクセスが処理効率に対するボトルネックとなり、細粒化に伴うメモリー・アクセスの増加は極めて重大である。

また、CPU間の足並みをそろえる同期処理もタスクの増加に従って多くなる。それに加えて近年、CPU性能の向上とともに演算時間は著しく短くなっており、演算単位の並列処理においては従来のように同期処理を無視することはできない。

これに対し著者らは、同期処理時間を共有メモリー・アクセスのバス調停期間中に行う効率的なハードウェアを提案し⁽⁶⁾、演算レベルの並列処理でも同期処理時間が無視できることを確認している。従って、このようなハードウェアの使用を前提とすれば、前者の共有メモリー・アクセスを最小化するスケジューリングを実現することで、ハードウェア的な問題がクリアできたことになる。

〈2・2〉 スケジューリングの問題点 タスクには入力データ転送と、演算結果を他のタスクへ渡す出力データ転送が存在する。これは各々のタスクがどのような順序で処理されても解を保証するため、タスクのデータはすべてメモリーに吐き出される形で記述されている。これにより、アキュムレータないしローカルメモリーに値が残っているにもかかわらず、次のタスクは共有メモリーから同じ結果をとってくるむだを行うことになる。

粗粒度並列処理⁽⁹⁾では、このような時間はタスク長に比べ十分に小さいとして無視してきたが、タスク長の極めて小さい細粒度並列処理でこれを無視することは問題が多い。従って、このような冗長なデータ転送に対する最適化を考えなければならない。しかしながら、これはNP困難なスケジューリング問題を更に難しくすることを意味する。

一方、細粒化に伴うタスク数の増加は、スケジューリング問題を組合せ規模で難しくさせる。従って、このような問題では一般に、最適解を求めようとすることは不可能であり、探索の打ちきりを前提としなければならない。こうした場合、探索の早期にいかにより良い候補を得るかが重要な問題となる。

分枝限定法は、様々なスケジューリング・アプローチの

なかでも、先天的な情報を効果的に利用できる効率的なスケジューラとして知られている。分枝限定法の探索効率を向上させるには、以下の2点が重要である。

(1) より高精度な下限関数による限定操作を行うこと。

(2) より良い上限値をできるだけ探索早期に得ること。

そこでここでは、(1)の問題に対し、(i)段取り時間を考慮した下限関数を提案する。また、(2)の問題に対しては、(ii)段取り時間とCPUの非実働時間による二つの上限値の併用と、(iii)ヒューリスティック順序を優先する探索法を提案する。

(2)の問題に対して、(iii)は特に重要であるが、これに必要なヒューリスティック順序は一意的に決定できない。特に細粒度の並列処理では、問題の規模が大きくなっていることに加え、これまで良いとされていた最遅開始時刻優先則⁽¹⁰⁾がそのまま適用できないため、一つのヒューリスティックに限定することは問題が多い。従ってこのような場合には、系統的シャフリングを行う複数探索則⁽¹¹⁾の効果が期待できる。

3. スケジューリング問題

本章では、段取り時間を考慮したスケジューリング問題を定式化し、その計算複雑度について考察する。

〈3・1〉 段取り時間とその表現 先行制約のある二つのタスクを同一プロセッサに連続して割り当てるときは、前の演算結果を他のプロセッサに転送する必要がないので、不要な転送を省くことを考える。ここでは、このようなデータ転送に伴う処理時間の差分を、真のタスク処理との差として区別する。また、これをその段取りを行うという意味で「段取り時間」と呼ぶ。なお、命令パイプラインや演算パイプラインをもつCPUなどでは、データのパイプによる処理時間と、共有メモリーのアクセスに伴う時間の差を段取り時間として記述することができる。

段取り時間を計算機で扱いやすい形で表すため、これを行列 $C_{i,j}$ の形で記述する。例えば、タスク間に先行制約のあるタスク i とタスク j が同一プロセッサに割り当てられた場合、タスク i の結果はタスク j の被演算項なので、段取り時間はデータをロードする分だけ小さくなる。しかし、先行制約のないタスク i とタスク k では、データを共有メモリーからロードしなくてはならないため、段取り時間はさきのものより大きくなる。このとき、データをロードする時間を t_{ld} すれば、 $C_{i,j}=0$ 、 $C_{i,k}=t_{ld}$ と書くことができる。

なお、ここで扱うモデルは「段取り時間が同一プロセッサ内の前後のタスクの組合せで決まる」と仮定している。

〈3・2〉 問題の定式化 以上の問題を定式化すると、以下のようになる。

「各々処理時間 t_i が異なり、 n 個のタスクからなる非分割(nonpreemptive)なタスク集合 N が、一般形状の先行

タスクをもち、かつ各タスクがその組合せにより異なる段取り時間 $C_{i,j}$ をもつという条件のもとで、能力の等しい m 台の機械により並列処理される際、そのスケジュール長を最小にする問題」

上で述べたように、段取り時間は同一プロセッサ内の前後のタスクの組合せで決まる場合のみを扱う。従って本論文では、行列 $C_{i,j}$ はコンスタントである。

〈3・3〉 計算複雑度 ここでは、探索中の空間複雑度を低く保つ特徴をもつ深き優先の分枝限定法を用いることを考える。先天的な情報はノードセレクションに反映されており、これをもとにヒューリスティック順序で探索を行う。

このような問題では、探索木における各ノードの分枝の数は、各割当て時点における割当て可能タスク(レディタスク)を使用可能なプロセッサに割り当てる組合せの数だけある。このとき、あるノードにおけるレディタスクの数を n_{ready} とし、使用可能なプロセッサ数を m_{av} とすると、各割当て時点で考慮されるべきアイドルタスク数 n_{idle} は、 $m_{\text{av}}=m$ のとき、 $n_{\text{idle}}=m_{\text{av}}-1$ である。また、 $1 \leq m_{\text{av}} < m$ のとき、 $n_{\text{idle}}=m_{\text{av}}$ であるから、段取り時間を考慮しない場合には各ノードの分枝数は以下のようになる⁽¹⁰⁾。

$$n_{\text{branch}} = (n_{\text{ready}} + n_{\text{idle}}) C_{m_{\text{av}}} \quad \dots\dots\dots (1)$$

一方、段取り時間を考慮した場合は、割り当てようとするプロセッサの最後に割り当てられているタスクとの組合せも考慮しなければならないので、分枝数は次のようになる。

$$n'_{\text{branch}} = (n_{\text{ready}} + n_{\text{idle}}) P_{m_{\text{av}}} \quad \dots\dots\dots (2)$$

従って、段取り時間を考慮することで計算複雑度は、(1)式に比べ、

$$(n_{\text{ready}} + n_{\text{idle}}) P_{m_{\text{av}}} / (n_{\text{ready}} + n_{\text{idle}}) C_{m_{\text{av}}} = m_{\text{av}}! \quad \dots\dots\dots (3)$$

倍悪化する。これは、探索ステージ当たりの計算時間がその時点での割当て可能な CPU 数の階乗で悪化したことを意味している。

しかし、もともとこの種の問題では最適解の導出は不可能であり、探索の打ち切りを前提とすることになる。こうした場合、探索の簡素化により探索効率を上げることが重要となる。次章ではこの点を考慮したスケジューリング法を提案する。

4. 段取り時間を考慮したスケジューラ

〈4・1〉 前処理部 スケジューリングを行う前に次の前処理を行う。

$$C_{i,j} = C_{i,j} - p_i \quad (j=1, \dots, n) \quad \dots\dots\dots (4)$$

$$t_i = t_i + p_i \quad \dots\dots\dots (5)$$

$$C_{i,j} = C_{i,j} - p_i \quad (i=1, \dots, n) \quad \dots\dots\dots (6)$$

$$t_j = t_j + p_j \quad \dots\dots\dots (7)$$

ここに、 p_i 、 p_j は行および列中の最小値であり、次のように計算できる。

$$p_i = \min_{j \in N} C_{i,j} \quad \dots\dots\dots (8)$$

$$p_j = \min_{i \in N} C_{i,j} \quad \dots\dots\dots (9)$$

ただし、 N は処理すべきタスクの集合、 n はその数を表す。この操作によって、 $C_{i,j}$ の行と列に少なくとも一つの 0 が存在することになる。

次に、このようにして変更したタスク集合 N に対し、出口ノードまでの距離 l_i を計算し、その大きい順にタスク番号をリナンバリングする。

〈4・2〉 限定操作 現在の割当て時刻 t_s をもとに最終的な下限推定値が計算できれば、現在の上限値を用いてより多くの部分問題を限定することが可能となり、探索の効率を上げることができる。ここでは、二つの上限値 U_t と U_{idle} を考える。 U_t は従来法と同様、暫定解のスケジュール長で、 U_{idle} は暫定解における機械の非実働時間と段取り時間の総和である。これらと下記の下限関数 t_{cr} 、 t_{div} 、 t_{cost} をもとに次のような限定操作を行う。

以下の条件のうちどちらかが満たされたとき、分枝限定法はそれより下の部分問題を考慮しない(バックトラック)を行う。

$$U_t \leq \max(t_{\text{cr}}, t_{\text{div}}) \quad \dots\dots\dots (10)$$

$$U_{\text{idle}} \leq t_{\text{ip}} + t_{\text{cost}} \quad \dots\dots\dots (11)$$

ここで、 t_{ip} は現時点までの機械の非実働時間と段取り時間の合計である。また、まだ割り当てていないタスクの集合を $I(\pi_a)$ とすると、出口ノードまでの距離をもとにした下限関数 t_{cr} は次のようになる⁽¹²⁾⁽¹³⁾。

$$t_{\text{cr}}(\pi_a) = \max_{i \in I(\pi_a)} l_i + t_s \quad \dots\dots\dots (12)$$

一方、 $I(\pi_a)$ の処理時間の総和を機械の数 m で割ったものは段取り時間を考慮した下限推定値を与え、以下のようになる。

$$t_{\text{div}}(\pi_a) = \lceil (\sum_{i \in I(\pi_a)} t_i + t_{\text{cost}}) / m \rceil + t_s \quad \dots\dots\dots (13)$$

ここで、 t_{cost} は $I(\pi_a)$ の段取り時間の下限推定値であり、以下のように計算できる。

$$t_{\text{cost}} = \max(t_{\text{row}}, t_{\text{col}}) \quad \dots\dots\dots (14)$$

$$t_{\text{row}} = \sum_{k=1}^{n_{\text{av}} - (m - m_{\text{av}})} \left\{ \min_{i \in I(\pi_a)} \left(\min_{i \in I(\pi_a), j \in R} C_{i,j} \right) \right\} + \sum_{k=1}^{m - m_{\text{av}}} \min_{i \in I(\pi_a)} (C_{i,n}) \quad \dots\dots\dots (15)$$

$$t_{\text{col}} = \sum_{k=1}^{n_{\text{av}} - (m - m_{\text{av}})} \left\{ \min_{j \in R} \left(\min_{i \in I(\pi_a), j \in R} C_{i,j} \right) \right\} + \sum_{k=1}^{m - m_{\text{av}}} \min_{i \in I(\pi_a)} (C_{i,n}) \quad \dots\dots\dots (16)$$

ただし、 R は未終了タスクの集合であり、既に割り当てたもののうち後ろの段取り時間がまだ決まっていないものを含む。また、 r はそのタスクの数である。

またここで、集合 X において k 番目に小さい要素を $\min^k(X)$ と定義した。すなわち $\min(X) = \min^1 X$ となる。

〈4・3〉 ノードセレクション 分枝限定法の多くは、探索木の左にできるだけ良い解を集める意味で、先天的な情報(確実な情報ではない意味でヒューリスティック順序

図 1 ノードレベル優先法の探索過程
Fig.1. Searching by the node-level precedence method.

と呼ばれている)を用いたノードセレクションが行われる。例えば笠原らは、CP/MISF 順序を深さ優先の分枝限定法に適用した DF/IHS を提案し⁽¹⁰⁾、その有効性を確認している。

ノードセレクションの方法は、探索木の各ノードにある分枝をヒューリスティック順序でソートし、番号の若い順に選ぶことで行うことができる。しかしながら、深さ優先の分枝限定法はバックトラックが下から進むため、探索を打ち切った場合には、上方ノードのヒューリスティック値の良いものの考慮が行われれないという問題がある⁽¹¹⁾。

そこで、以下に示すノードセレクションを提案する。

まずノード番号 (L_i) は各ノードごとに 0 から始まる整数で付ける。そしてノードセレクションのたびにこのノード番号の和をとる。ここではこれをノードレベル (L_n) と呼ぶ。そして、その和がある設定値(これを探索レベル L_s と呼ぶ)を超えたとき、限定操作を行う。これにより、空間複雑度の小さなノードレベル優先の分枝限定法を構築することができ、探索の打ち切りを前提とした深さ優先での上方ノードの再考慮の問題を解決することができる。図 1 に各探索レベルにおける探索過程の例を示す。図より、提案したノードセレクションの効果が確認でき、探索レベルの小さいものほど早期に良い上限値を与えることがわかる。これは、ノードレベルに制限を付けたことでヒューリスティック順序に則した探索が行えるためで、探索の打ち切りを前提とした場合は特に効果的である。また、早期に良い上限値を得ることができることから、これを並列探索の一部に用いることで、並列型分枝限定法の効率を上げることができる。

しかしながら、 L_s を小さく設定すると探索木を読み切る。このときは L_s を大きくとりなおして再度探索することになるが、上記のような単純なアルゴリズムでは、同じ探索を再び行うむだが生じる。そこでノードセレクションを以下のアルゴリズムによって行うことにする。

- (1) $L_n = 0$
- (2) 探索木の各ノードセレクションでは、 $L_n + L_i$ が

図 2 分枝の選択順序
Fig.2. Order selection of the branches.

探索レベル (L_s) と同じか一番近い小さな数のノードが選ばれる。そして、バックトラックはこれより小さいノードに向かって行われる。

- (3) $L_n = L_n + L_i$ を計算し、暫定解に到達するまで、(2) を再帰的に繰り返す。

- (4) 暫定解が得られたとき L_n が L_s に等しいことを確認する。等しくなければ、バックトラックする。

例えば図 2 において、 $L_s = 1$ の探索は a, b, c の順に行われる。なお、ノード番号は 0 から始めることで、ノードの深さによらないノードレベルの定義が可能となっている。

以上の処理を L_s を 0 から始めて、探索木を読み切るときにこれを一つずつ増やしながら探索してゆくことで、重複探索を行うことなしに、ヒューリスティック順序に則した探索を行うことができる。

〈4・4〉 複数領域の探索 ところで、複数探索則の並列処理と系統的なシャフリングによっても、分枝限定法の探索効率を促進することができる⁽¹¹⁾。特にここでは、ノードセレクションで用いるヒューリスティック則に段取りを考慮していないため、これまで良いとされてきた「最遅開始時刻優先探索則」の効果は必ずしも期待できない。また、問題の規模の大きいことから、一つのヒューリスティックへの固執は、従来の段取り時間を考慮しない場合に比べ問題となる。そこでここでは、以下のようなヒューリスティック順序の分枝限定法の並列探索を試みる。母体となるスケジューラはクリチカルパス法を基にした、リストスケジューラを拡張したものであり、笠原らの DF/IHS⁽¹⁰⁾ と基本的に同じである。ただし、文献(11)に示すように、探索早期では探索則を頻繁に切り換え、探索が進むにつれてその間隔を広げてゆく系統的シャフルを行っている。

- (1) 最小最遅開始時刻最大後続タスク数優先規則
- (2) 最小最遅開始時刻優先規則
- (3) 優先順位を付けない(単純な CP 法であり、リストは乱数で選ばれる)

これらはすべて深さ優先の分枝限定法で解く。系統的シャフリングは、文献(11)と同様、(1), (2) に対してはシャフリング間隔の更新度数 $\eta = 1.15$ で行う。そして、(3) は悪い解が得られる確率が高いので、短い間隔でしば

ば行う。

ここでは更に、これら三つの探索則に<4・3>節で提案した手法を加える。すなわち、

(4) (1)をノードレベル優先の分枝限定法で解くを加えた計四つの複数探索を行う。

なお、(4)はシャプリングは行わず L_s を 0 から順に探索してゆく。

5. 実験

<5・1> 段取り時間の影響 提案したスケジューラを用い、乱数で発生した3種類のタスクグラフ(タスク数30, 50, 90)を段取り時間とタスク長の比を変えて解き、改善率を評価した(図3)。

ここで、改善率 IM は、次のように定める。

$$IM = \frac{PN - PC}{PC} \times 100 \quad (\%) \quad \dots\dots\dots (17)$$

ここに、 PC : 段取り時間を考慮したときの暫定解、 PN : 段取り時間を考慮しないときの暫定解
グラフの各点は各々、50個のタスクグラフの平均でとった。従って、実験は全体として600サンプルである。なお、ここでは文献(11)と同様の接続形態(プログラムはUNIXで実現され、各々のCPUはEther-netによる疎結合並列処理方式)の4台のワークステーション Sparc-Station(sun)を用い、探索時間は各々2分間(総探索時間では8分)とした。プロセス間の通信はソケットを用いて行っている。これより、段取り時間を考慮した場合のスケジューリング誤差は、タスク処理時間に対する段取り時間の比にほぼ比例し、例えば、段取り時間がタスク長の20%を占めたとき、8~12%のスケジューリング誤差が生じていることがわかる。

<5・2> 実問題への適用 次に文献(14)の3軸マニピュレータ逆動力計算の並列処理を行ってみた。ロボットモーションのシミュレーションのためにはLagrange方程式やNewton-Euler法を用いる方法が提案されているが、ここでは、比較的計算複雑度の少ないNewton-Euler法

で行う。

実験には、「同期処理をアービタが行う共有メモリ型マルチプロセッサシステム」⁽¹⁵⁾を用いた。これはDSP(TMS 320 C 25)によりインプリメントされている。各タスクの実行時間は乗算12 clock, 加減算8 clock, 段取り時間12 clockである。このように、段取り時間が演算時間に比べ無視できない大きさであり、段取り時間を考慮したスケジューリングの効果が期待できる。

ここでは次に示すような条件のもとで、データ転送時間が省略できるとして段取り時間行列を作成した。

(1) タスク(task j)のデータ参照が、同一プロセッサに直前に割り当てられたタスク(task i)の代入データと同一変数であるとき、task j のデータ参照処理は省略できる。

(2) Task i の代入変数を参照する後続タスクがただ一つTask i だけであるとき、Task i のデータ代入処理とTask j のデータ参照は省略できる。

(2)により、式を分割した際にできる一時変数が省略でき(1回しかアクセスされない変数はレジスタに配置され)、効率的なコード生成を行うことができる。

実験は次の三つの方式に対して行った。

(L) 各リンクごとにタスク分割する従来の方法($n=16$)を文献(10)のDF/IHSによって解く。

(F1) 演算単位のタスクで段取り時間を考慮したもの($n=187$)を提案手法によって解く。

(F2) F1と同じタスク集合を用い、段取り時間を無視してスケジューリングした(以下、F20と記す)後、必要な段取り時間を付け加えたもの。ただし、F20は文献(11)のアルゴリズムに<4・3>節で提案したノードセレクションを付加したスケジューラを用いた。

今回は扱う問題の特殊性から他の手法との比較は上記F20に限るが、これは文献(11)で有効性が確認されている。文献(11)では広く応用されているDF/IHSやCP/MISFとの比較もあることから、限られた比較ではあるが、これとの比較によって提案手法の有効性を示すことが

図3 段取り時間の大きさと改善率

Fig. 3. Ratio of Preparation time and improvement ratio.

図4 3軸マニピュレータの並列処理

Fig. 4. Parallel processing for a 3DOF manipulator.

できると考える。

結果を図4に示す。提案手法では、予想どおり探索木を読み切ることができなかったが、(L)およびF20はいずれも、すべてのCPU数において最適解を導出した。しかし、F20はF1より極めて良い解を得たようにみえるが、実際には無視した段取り時間を含めることになるため、F2と悪くなっている。このように段取りを無視してスケジューリングすると、たとえ良いスケジュール結果を得ても、実際にはCPU数によらず34.6~36.3%といった大きな誤差を含んでおり、近年問題とされている数%のスケジュール精度の改善は事実上無意味である。このような状況が発生するため、細粒度並列処理は一般に困難なものとなっていた。

一方、提案手法F1では、段取時間を考慮することによって、スケジューリング誤差をなくすことができ、高精度な細粒度並列処理を実現することができた。解精度の推定はできないが、F2と比べて良好な結果が得られている。F2との差はCPU数が小さいほど大きく、1台で24.2%、2台で16.1%であった。なお、差はCPU数が増えるほど少なくなっていくように見えるが、すべてのCPUの使用時間の合計で比べれば、減少してない。

また期待したとおり、細粒化によってクリチカルパス長が短縮し、CPU数 m を増やすことができれば、処理時間の大幅な改善を行えることが確認できた。

以上のことから、最適なスケジューリングが行えなくても、細粒化によるクリチカルパス長の短縮による効果が期待できるのなら、このような並列処理を行う意味があると考えられる。

6. おわりに

演算レベルまで細かく分割したタスクを、段取り時間行列を用いてスケジュールする方法について述べた。これによって、いったん細かく分割されたタスクが、データフローを考慮しながら最適組合せによって結合され、以下に示す効果が確認できた。

(1) 細粒度並列処理において従来生じていた冗長な共有メモリー・アクセスを減らすことができた。

(2) 効率的な下限関数や探索順序の検討によって、従来あった大きなスケジューリング誤差を軽減でき、細粒度並列処理の実現の可能性を示した。

(3) 細粒化によるクリチカルパス長の短縮によって、CPU数が増やせるならば大幅な処理時間の改善が行えることが確認できた。

乱数で発生したタスクグラフに適用した結果も良好で、

(4) 一般的に段取り時間を考慮した場合のスケジューリング誤差は、タスク処理時間に対する段取時間の比にほぼ比例し、例えば、段取時間がタスク長の20%を占めたとき、8~12%のスケジューリング誤差が発生することなどもわかった。

なお本手法では、共有メモリー・アクセスは必要最小限

となることから、共有メモリー型マルチプロセッサシステムにおけるバスボトルネックの問題を緩和するメリットもある。

また、実問題としてマニピュレータの逆動力学計算に適用した結果、細粒化によりクリチカルパス長が短縮でき、段取時間を考慮しない場合のスケジューリング誤差(約34.6~36.3%)をなくすことができた。

なお、ここでは文献(15)で提案した特別なハードウェアの使用を前提とし、段取時間の中に含まれるプロセッサ間の同期処理時間や通信時間の動的な変動については扱わなかった。そのため、トランスピュータのようなメッセージパッシング方式におけるデータ伝搬遅れ時間の考慮などが今後の課題である。

(平成5年3月31日受付, 同5年9月24日再受付)

文 献

- (1) 大山:「組合せ問題の計算上の複雑さについて」, オペレーションズ・リサーチ, No. 7, 427 (昭54)
- (2) S. Kirkpatrick, C.D. Gelatt & Jr., M.P. Vecchi: "Optimization by Simulated Annealing", *Science*, **220**, No. 4598, 671 (1983)
- (3) 西川・玉置:「ジョブショップ・スケジューリング問題に対する遺伝アルゴリズムの一構成法」, 計測自動制御学論, **27**, No. 5 (平3)
- (4) B. Kruatrachve & T. Lewis: "Grain Size Determination for Parallel Processing", *IEEE Software*, **1**, 23 (1988)
- (5) M.W. Walker & D.E. Orin: "Efficient dynamic computer simulation of robotic mechanisms", in Proc. Jt. Au-tom. control conf (1981)
- (6) J.Y.S. Luh, M.W. Walker & R.P. C. Paul: "On-Line computational scheme for mechanical manipulators", *ASME J. Dynamic Syst., Meas., Contr.*, **102**, 69 (1980)
- (7) 橋本:「逆ダイナミクス of 新しい並列計算法」, 計測と制御, **30**, No. 5, 406 (平3)
- (8) 近藤・小林:「演算並列処理のための同期制御バスアービタ」, 電学論C, **109**, 57 (平元-2)
- (9) R.G. Babb II: "Parallel Processing with Large-Grain Data Flow Techniques", *IEEE Comput.*, No. 7, 55 (1984)
- (10) H. Kasahara & S. Narita: "Practical multiprocessor scheduling algorithms for efficient parallel processing", *IEEE Trans. Comput.*, **C-33**, No. 11, 1023 (1984)
- (11) 近藤・小林・原・武平:「系統的シャフルを用いた並列型分枝限定法—マルチプロセッサタスクスケジューリングへの応用—」, 電学論C, **113**, 101 (平5-3)
- (12) T.C. Ho: "Parallel Sequencing and Assembly Line Problem", *Oper. Res.*, **9**, 841 (1961)
- (13) E.B. Fernandez, & B. Bussell: "Bounds on the Number of Processors and Time for Multiprocessor Optimal Schedules", *IEEE Trans. Comput.*, **22**, No. 8, 745 (1973)
- (14) 吉川:「ロボット制御基礎論」コンピュータ制御機械システムシリーズ10, (昭63) コロナ社
- (15) H. Kondoh, T. Hosoi, et al.: "Parallel Processing of Finely-Grained Tasks: Arbitrating Synchronizer and Parallel Scheduler", *IEEE IECON'91*, 2, 1425 (1991)

近藤 仁 志 (正員) 1985年長岡技術科学大学大学院修士課程修了。同年徳山工業高等専門学校助手。同講師を経て、1990年東京工業大学助手。マルチプロセッサシステム、スケジューリングなどの研究に従事。電子情報通信学会、計測自動制御学会会員。

細井隆史

(非会員) 1993年東京工業大学大学院修士課程修了。同年ソニー(株)入社。マルチプロセッサシステム、スケジューリング、デジタル信号処理に関する研究に従事。

原辰次

(非会員) 1976年東京工業大学大学院修士課程修了。同年日本電信電話入社。1980年長岡技術科学大学助手。1984年東京工業大学助教授、1992年同教授。1988年～1989年カナダWaterloo大学客員研究員。工学博士。ロバスト制御、デジタル制御に関する研究に従事。計測自動制御学会、情報処理学会、電子情報通信学会、IEEE会員。