

フローショップスケジューリングにおける総所要時間 最小化問題に対する並列アルゴリズムとnCUBE2上の実現

岡本秀輔 渡辺一衛 飯塚 肇

成蹊大学大学院工学研究科 情報処理専攻
東京都武蔵野市吉祥寺北町3-3-1

あらし 本報告では、 n ジョブ、 m 機械のフローショップスケジューリング問題に対する並列最適化アルゴリズムを提案する。このアルゴリズムは、フローショップ問題に対する分枝限定法を並列化し、更に、あるサイズより小さな部分問題を全探索することにより探索の効率化を実現したものである。アルゴリズムの性能は、分散記憶型マルチプロセッサnCUBE2上での並列実行と分枝限定法の逐次実行との比較により評価した。この結果、32台のプロセッサを用いた場合、逐次の実行時間が1分より長い問題において平均して32倍以上の速度向上が確かめられた。また、逐次の実行時間が比較的短い問題においても、減速することなく解を与えることが確かめられ、アルゴリズムの有効性を確認した。

和文キーワード 並列アルゴリズム、フローショップスケジューリング、分枝限定法、nCUBE2

A New Parallel Algorithm for the n -job, m -machine Flow-shop Scheduling Problem and its Implementation on nCUBE2

Shusuke OKAMOTO Ichie WATANABE Hajime IIZUKA

Department of Information Sciences, Seikei University
3-3-1 Kichijoji-Kitamachi, Musashino-shi, Tokyo

Abstract

This paper describes a new parallel algorithm for solving n -job, m -machine flow-shop problems. The algorithm is basically a parallelization of the usual branch-and-bound method, but it also takes advantage of all search method, to keep high the efficiency of parallel processing, when the sub-problem becomes smaller than certain size. It is shown that its implementation on nCUBE2 gives very good performance characteristics.

英文 key words Parallel algorithm, Flow-shop scheduling, Branch-and-bound technique, nCUBE2

1 はじめに

フローショップ過程は、 $m_1, m_2, \dots, m_m$ と番号付けされた m 台の機械と n 個の異なるジョブからなる。各ジョブは図1のように機械の番号順に一度ずつ処理される。各機械におけるジョブの処理時間を与えられており、機械ごとのジョブの処理順序は同一であるという条件を加える。

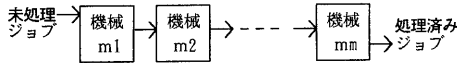


図1 ピュア・フローショップ

このような前提において、処理されるジョブの順序を決めてやることにより、総所要時間（以下 makespan とする）、平均遅れ時間、スループット等を最小化するのが（ピュア）フローショップ問題であり、スケジューリングの分野において古くから扱われている問題である。この問題は、ジョブ数の階乗個の可能解が存在するために、ジョブ数の多い問題に対しては効率の良い探索が必要となる。

今回扱うフローショップ問題を正確に条件付けると以下ようになる。：

- (1) 時間 0 において n 個の全てのジョブが処理可能である。
- (2) 各ジョブは m 段階の処理を必要とし、その各々は異なった機械で処理される。
- (3) 各処理のセットアップ時間は処理される順序に無関係で処理時間に含まれる。
- (4) 各ジョブは $m_1, m_2, \dots, m_m$ の順に機械を通して処理されていく。
- (5) 処理されるジョブの順序は各機械間で共通である。
- (6) m 台の機械はそれぞれ連続して使用できる。
- (7) 個々の処理について割り込みは許されない。

次に makespan の最小化問題について考える。一般に、各機械におけるジョブの処理時間はジョブ番号を $j_1, j_2, \dots, j_n$ と表すと図2のようなテーブルで与えられる。

	j_1	j_2	j_3
m_1	2	2	5
m_2	3	3	2
m_3	2	3	2

図2 問題テーブル

この問題に対して例えば (j_2, j_3, j_1) というスケジュールを考えると、そのガントチャートは図3のようになりこれから makespan は 14 と分かる。

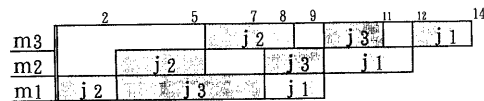


図3 ガントチャート

これを実際に計算させるには、このガントチャートを作るときと同じ手順で以下のようなステップになる。

- ① 変数 v_1, v_2, v_3 を用意し 0 に初期化する。

- ② 全てのジョブを順次スケジュール順に対象ジョブとして、③④⑤を繰り返す。
- ③ v_1 に対象ジョブの m_1 での値を加える。
- ④ v_1 と v_2 の大きい方に対象ジョブの m_2 の値を加えた数を v_2 に代入する。
- ⑤ v_2 と v_3 の大きい方に対象ジョブの m_3 の値を加えた数を v_3 に代入する。

こうして最後に得られた v_3 の値がこのスケジュールの makespan となる。最小値を求めるには全てのスケジュールについて計算すればよい。この計算自体はそれほど複雑ではないが、先も述べたようにジョブ数が増えると計算するスケジュールの数はその階乗通りになるために計算量が爆発的に増大してしまう。

makespan の最小化アルゴリズムとして、Ignall と Schrage の分枝限定法がよく知られている。このアルゴリズムにおける下界値の計算は、その時点までに選択された部分スケジュールと各機械における未選択ジョブを用いて以下のように計算される。

現在までに決定された部分スケジュールを σ とし、これに含まれないジョブ集合を σ' とする。 σ における $m_1, m_2, \dots, m_m$ の完了時刻をそれぞれ $q_1, q_2, \dots, q_m$ とする。また、 t_{ji} をジョブ j の機械 i における処理時間とする。

まず m_1 について考える。全ての処理が終了した時の m_1 の完了時刻は q_1 に

$$\sum_{j \in \sigma'} t_{j1}$$

を加えたものになる。さらに m_1 において一番最後に処理されるジョブ k を考える。ジョブ k が m_1 での処理が完了した後、全ての処理が終わるには少なくとも $(tk_2 + tk_3 + \dots + tk_m)$ だけかかる。したがって下界値の一つとして、

$$b_1 = q_1 + \sum_{j \in \sigma'} t_{j1} + \min_{j \in \sigma'} \{t_{j2} + t_{j3} + \dots + t_{jm}\}$$

を算出することができる（図4）。

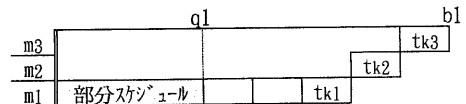


図4 b_1 の計算

m_2 についても同様に考え、

$$b_2 = q_2 + \sum_{j \in \sigma'} t_{j2} + \min_{j \in \sigma'} \{t_{j3} + \dots + t_{jm}\}$$

とすることができ、 m_3 から m_m についても同様に $b_3, b_4, \dots, b_m$ をそれぞれ計算する。これらの値の中での最大値つまり

$$B = \max \{b_1, b_2, \dots, b_m\}$$

が限定操作の下界値となる。

このアルゴリズムでは、下界値の計算が複雑になっている反面かなり効率良く探索を行うことができる。しかし、ジョブ数が 10 程度の問題からは、問題により探索範囲が広くなり短時間では終了しない場合が出てきてしまう。

本研究では、この分枝限定法を並列化することにより、逐次アルゴリズムで高速に解ける問題をそれよりも遅くならず解き、さらに探索に長時間を要する問題を 1 / (プロセッサ数) の時間内

で解くアルゴリズムの開発を行いnCUBE2上でこのアルゴリズムの評価を行った。

2 nCUBE2の概要

ここでは今回の測定に用いたnCUBE2について述べる。提案するアルゴリズムは特にnCUBE2のような分散記憶型の計算機専用ではないが、後の評価の参考のためにここで紹介する。

2.1 ハードウェア

nCUBE2はプロセッサ間の結合にハイパーキューブネットワークを採用した分散記憶型のマルチプロセッサである。ハイパーキューブトポロジーは多次元の立方体として構成され、その立方体の各頂点にCPUと主記憶から成る計算部分(CE:コンピュティンク・エレメント)が配置されている。CEは最大8192個まで実装可能である。CE間の通信はノード間リンクを通してハードウェアルーティングによる高速の直列転送で行われ(wormhole)、通信速度は公称値で2.22MB/secである。各CE中のプロセッサは、DEC社のVAXマシンと類似な命令セットを持ったカスタムLSIを用いており、単体性能は7.5MIPS、3.5MFLOPS(32ビット)、2.4MFLOPS(64ビット)である。

今回の測定では5次元(32プロセッサ)のものを用了。

2.2 ソフトウェア

2.2.1 ノードOS

nCUBE2はVERTEXというノードOSを各ノードが持っている。VERTEXは、UNIXワークステーションをフロントエンドとして、マルチタスクをサポートするとともに、Sunワークステーション上のファイルシステムや標準入出力をユーザが特にVERTEXを意識することなく使えるようにしている。

またVERTEXはプログラムに対してハイパーキューブの次元を、論理的な次元として提供している。このため例えば5次元(32個)のマシン構成であれば、プログラムとしてはマシンが0~5次元のどの次元であると仮定しても良い。プログラム中では、この論理ハイパーキューブ次元でのプロセッサ番号を使うことになる。マルチタスクでのプロセスを指定する時はこのプロセッサ番号とそのプロセッサでのプロセス番号の二つを用いる。プログラムが使用するハイパーキューブ次元は、プログラムのロード時に決定される。

2.2.2 プログラミング言語

nCUBE2はプログラミング言語として、ANSI規格のFORTRAN-77およびC言語そしてアセンブラをサポートしている。これらの言語はホストマシンであるSunワークステーション上でnCUBE2用にクロスコンパイル(アセンブル)されるので、ファイル管理やエディタの使用などは、全てUNIX上で行うことができる。また、これらの言語は、構文的には並列処理用の拡張が行われておらず、並列処理に必要な通信等は全てライブラリ関数を使用して行う。

3 並列化

3.1 方針

一般に探索問題を並列化した場合、逐次アルゴリズムにおいて限定操作により除外される部分問題を無駄に探索する可能性が生じる。これはある程度は避けられないが、部分問題を分割しすぎて逐次アルゴリズムより遅くなってしまう最悪の事

態(減速)を招き並列化の意味がなくなる。また、これとは逆に解く問題によっては探索空間が狭まりプロセッサの台数倍以上の速度向上が起こりうる(スーパー・リニア・スピードアップ)。スーパー・リニア・スピードアップは非常に望ましい現象ではあるが、この現象の速度向上率の値のみを追究してその発生頻度を無視してしまつてはアルゴリズムの正しい評価が行えない。極端な場合、例えばN台のプロセッサを用いて多数の問題を解かせ、8~9割りが減速で、残りがスーパー・リニア・スピードアップであったとする。このアルゴリズムの性能は、スーパー・リニア・スピードアップの時の速度向上率の値により、全体の平均値としてN倍に近い値と計算することは可能であるかもしれない。しかし、このアルゴリズムを実際に使用したとき、その印象は逐次より遅いものとなってしまうであろう。またこのように良い場合の頻度が低いならば、一台のプロセッサで並列探索をシミュレートすれば良いということにもなる。したがって、減速、スーパー・リニア・スピードアップおよびそれらの中間の三つに分けた場合、それぞれの平均の値の他に、それらの発生頻度がどうかという点も解法の評価をする上で重要である。

次に、分枝限定法を用いる場合その探索法を選択しなければならない。フローショップ問題の場合、先にも述べたがジョブ数の増加により探索空間が莫大に増加するために、メモリ使用の面を考えて深さ優先探索(以下DFと記述する)を選ぶこととした。DFでは問題を読み込んだ時点(ジョブ数が分かった時点)で、使用するメモリの上限を計算することができ、さらに暫定解を計算するためには必ずこの上限だけは必要となる。そこでプログラムの初めに必要な量だけのメモリを獲得することにより、C言語のmallocやfreeといったメモリ管理にその速度が依存する関数の使用を最少に押えることができる。これは特に逐次アルゴリズムとの比較を考えたときに大きな意味を持ってくる。動的メモリ管理の効率があり良くなければ、様々なサイズのメモリブロックの獲得と開放を繰り返すうちに、獲得の時間が長くなる可能性が出てくる。この環境においてメモリの獲得と開放を繰り返して使う並列アルゴリズムと逐次アルゴリズムの比較をした場合、並列版と逐次版で探索範囲が同じでメモリの獲得と開放の延べ回数が同じだったとしても、逐次版の方がメモリ獲得に時間が掛かる可能性が高いために(並列版では1台あたり1/N回ですむ)並列版のアルゴリズムの性能がより良いかのようにみえてしまう。

その他の設計方針としては単純な構造のアルゴリズムとした。これは、種々様々に存在する並列処理計算機上(粗粒度のMIMDマシンに限定されるが)で容易に実現できることを意味し、アルゴリズムの利用度を上げるものである。

3.2 概要

プログラムはマスタに対する複数のワーカといった形で連結されたプロセスにより探索が行われる。プログラムがロードされると、まずマスタが問題を読み込み全てのワーカにそれを放送して送る。マスタはワーカが探索中であるか問題の割

当て待ちであるかを把握して、待ち状態のワーカー手持ちの部分問題を割当てて。待ち状態のワーカーがいるが部分問題を持っていないときは、探索中のワーカーへメッセージを送り問題を返して貰う。またその他に、ワーカーから送られてくる暫定解の候補を処理し、その時点で一番良い暫定解を放送する。探索の終了はマスタが判断する。

ワーカーは最初の自己スケジュールまたはマスタからの割当てにより探索を開始する。探索は基本的にはDFにより行いが残りジョブが4ジョブ以下の部分問題は全探索を行う。また、暫定解の候補が見つかるたびにマスタへそれを送る。

以下このアルゴリズムをPDFALL(Parallel Depth First and ALL search)と呼ぶ。

3.3 データ構造と通信の種類

3.3.1 部分問題配列

PDFALLにおいて、部分問題はマスタ・ワーカー間で転送されるので、図5のような固定長の配列で表現してアクセス方法を単純化している。

LB	level	q	schedule
----	-------	---	----------

図5 部分問題配列

ここでLBはこの部分問題の下界値であり、levelはスケジュールされたジョブの数でそれぞれ4バイトを占める。qは図4で述べた変数の並びつまり選択されたスケジュールでの各機械の完了時刻であり、(機械数×4)バイトである。scheduleはlevel個の選択されたスケジュールである。ただしサイズは、(ジョブ数-1)×4バイトが確保されている。

構造体を用いずに配列を用いたのは、qとscheduleが解く問題により変わってくるためである。

3.3.2 問題の放送

解くべき問題は、図2を表現したファイルに入っている。nCUBE2ではどのプロセッサもファイルアクセスできるが、本研究ではマスタのみがファイルにアクセスできると仮定した。そこでマスタはプログラムの始めにファイルから問題を読み込み、放送してそれをワーカーに送る。問題の送信にはまず機械数とジョブ数を送り、ワーカーに必要メモリを確保させた後、問題データを送る。

3.3.3 再割当て要求と再割当て

ワーカーは探索すべき部分問題がなくなるとそれをマスタに伝え通信待ち状態になる。この時マスタにはそれを知らせるだけで特に何かデータを送る必要はない。実現では、nCUBE2の0バイト送信の機能を使っている。

再割当てはマスタから図5の形式のデータを割当てて要求をしているワーカーに送って行く。

3.3.4 部分問題返送要求と返送

マスタが再割当て要求を受けた時に割当てるべき部分問題を持っていなかったならば、探索中の全てのワーカーへ未探索の部分問題を返して貰うよう伝え、そのワーカーをおぼえておく。マスタはこれら全てのワーカーから部分問題の返送または再割当て要求がくるまで次の返送要求を出さない。この返送要求は再割当て要求と同様に伝えるだけでデータを送る必要はない。

要求を受けたワーカーは図5の形式で未探索の部分問題をマスタに送る。送るべき問題がない時は、そのまま探索を続ける。このようなワーカーはすぐに自分も探索すべき部分問題がなくなり再割当て要求をすることになる。

3.3.5 暫定解候補の送信と暫定解の放送

ワーカーは、探索中に実行可能解を見つけ、かつそれが自分の知っている暫定解より良かったならば図5の形式でその解をマスタに送り、それを新しい暫定解とする。

マスタはワーカーから暫定解の候補を受け取り、現在の暫定解と比較して良かったならば、暫定解の値(下限値)を放送する。

3.3.6 終了の通達

マスタが探索の終了を判断したとき終了の命令が放送される。

3.4 マスタの処理

3.4.1 問題の再割当て

マスタが保持する部分問題はLBつまり下界値の小さい順に整理されている。よってこのリストの順に再割当てが行われていく。このリストの要素は、後に述べる自己スケジューリングの対象にならずにマスタが見つけた部分問題(ジョブ数がワーカー数より多いときに起こる)か、ワーカーからの返送された部分問題である。前者はレベル1(探索木の一番上のレベルで先頭ジョブだけが決まった部分問題。ジョブ数だけ存在する。)まで決定された部分問題であり、後者はワーカーが返送するときに手持ちの中でレベルの低いほうから部分問題が選ばれる。よって再割当てでもプログラム全体としてレベルの低いほうから行われ、頻繁に再割当てが起こるのを防いでいる。

3.4.2 終了条件

終了条件は、マスタが判断する。全てのワーカーが待ち状態であり、かつ手持ちの部分問題および返送された問題がなく、さらに暫定解の候補をすべて受け取ったときとなる。

3.5 ワーカーの処理

3.5.1 自己スケジューリング

最初の割当てにおいて、ジョブ数分のワーカーは自分で探索領域を決め、マスタ側もそれらのワーカーが探索中であるとする。その方法は以下のようである。まず問題を受け取ったワーカーは、自分のワーカー番号と問題のジョブ数を比較し、自分の番号が小さければ探索を開始し、大きければマスタからの部分問題の割当てを待つ。探索を開始した全てのワーカーは、探索木の1番上のレベルのノードを展開する。展開したノードのうちワーカー1は1番下界値の良いもの、ワーカー2は2番目に下界値の良いものというようにして自分が探索すべき部分問題を決定し探索を行っていく。このようにすることにより、探索木の左側(下界値が良いノードほど木の左側に配置したとして)を探索するワーカーほど探索を速く進めることになる。解は木の左側に存在する可能性が高いので、このように木の左側に存在する暫定解の候補を先にマスタに送ることは価値がある。もし最初にマスタに送られた解が暫定解となれば、暫定解の放送が一回ですむなど通信量を大幅に減らすことができる。無駄な部分問題の探索も早い段階で減らすこ

とができる。またこの木の左側を先行する探索は逐次のDFと同じであり、DFが高速に解を発見するような探索木の場合、その解となりうる暫定解をほぼ最初の問題の放送にかかるだけの時間の遅れで発見することができる。そしてそこからの限定操作は $1 / (\text{ワカ数})$ の時間でできるために逐次のDFより遅くなる場合をかなり減らすことができる。

3.5.2全探索

逐次計算において分枝限定法は下界値の計算が複雑であるために、ジョブ数が少ない問題では全探索の方が速く解を見つけることができる。今回用いたプログラムにおいて、計算機が行う加算、比較、代入をそれぞれ等しく1単位時間かかると仮定して、逐次のDFと全探索の演算回数を比較してみると図6のようになる。これは10機械の場合で、DF maxはDFが最悪のケースとして全てのノードを探索した場合、DF minは最小ノードで最適解を見つけた場合、allは全探索の演算回数をそれぞれ表している。

10機械に限らず4ジョブでは全探索とDFのminがかなり近い値になるので、4ジョブ以下では全探索の方が性能が良いことになる。そこで予備調査として、DFに4ジョブの部分問題を全探索させるアルゴリズム(仮にDFallとする)を考えて、DFと比較してみた。DFallは仮想的なジョブj1~j4の4ジョブの可能なスケジュール24個をあらかじめ表として持ち、DFの部分問題が4ジョブになったところで、残った4ジョブをこのj1~j4に割当てて全探索するものである。この調査の結果、DFallはDFに比べて数%の速度向上が確認された。

このDFallを並列版のワカに組み込むことを考える。ワカの探索ではそれぞれの下界値の計算時間が短くなることはもちろん重要であるが、それよりも良い値を他のワカに反映させることにより無駄な探索を減らすことが更に重要である。そこで並列版では、全探索中に現在の暫定解より良い値が見つかったならばすぐにマスタにその値を送るようにした。このようにすることにより、のり24通りの計算において、最高でワカが普通にDFで探索する時間の24分の1でマスタに

暫定解の候補を送ることができる。また、計算する順番の関係で、24通りの始めの方で最良でない値をマスタに送ったのとしても、それにより他のワカの探索空間が小さくなる可能性は十分にある。ここで問題になるとすれば、最悪のケースとして24通りのほとんどの値をマスタに送ってしまい、通信量が増えすぎてしまう場合が出てくることである。しかし、確率的にこれはそれほど起こるわけではなく、またマスタに送られた値が全て放送されるわけではないので大きな問題にはならない。ただし、マスタ・ワカ型の分りきった欠点としての、ワカの数に対してマスタの処理能力が十分でなければならぬという制約に関する問題なので、ワカの数が多い場合には何等かの配慮が必要となってくるであろう。

4性能評価

本章では分散記憶型マルチプロセッサNCUBE2における処理時間の計測結果について述べる。計測には32プロセッサを用いて逐次のDFとの比較を行った。全探索の効果を調べるために、ワカの探索にDFをつかうPDFを実現し、同様の計算式を用いて比較する。

4.1問題のパラメタ

これまで述べてきたフローショップの分枝限定法は、ジョブ数、機械数、処理時間の範囲の3つのパラメタによって特徴付けられる。始めにも述べたがこの問題は可能解がジョブ数の階乗個あり、探索木の全末端ノードもこれと同数できる。つまりジョブ数が問題空間の大きさを決めるパラメタとなる。次に、機械数は各ノードの下界値の計算時間に関係する。図4のところで述べたが、b1の計算においてジョブkを決定するためには、未決定ジョブどうして残り機械の処理時間の総和を比べなければならない。また、これをb1からbmまで機械の数だけ計算しなければならない。よって機械数は総計算時間を大きく左右する。最後に各機械における処理時間の範囲であるが、これは不確定要素である。直観的には(ジョブ数×機械数)個の値に対して取りうる範囲がかなり大きければ可能解それぞれのmakespanも違う値ばかりとなるであろうから、最適解は探索木の左側による可能性が高い。また逆に小さければ今度はmakespanは近い値になるであろうから、最適解が探索木の左側によるとは限らず逐次の探索時間を長くなるであろう。しかし極端に範囲が小さければmakespanの値が同じものばかりになって探索が高速に終了するようになる。

今回のアルゴリズムの評価では以上のことを踏まえて、図7のような範囲で各50問づつ合計5000問の問題を解き実行時間を測定した。乱数にはCのrand()関数を用いた。

ジョブ数	5~14の10通り
機械数	5~14の10通り
処理時間の範囲	1~100の一樣整数乱数

図7 問題パラメタ

4.2実行時間分布

図8、図9、図10はそれぞれDF、PDF、PDFALLのジョブの変化による実行時間の分布を表したものの

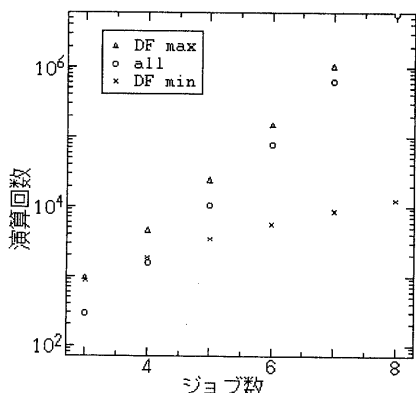


図6 10機械における演算回数の比較

である。時間枠が対数的にとられていることから、14ジョブにおけるDFはかなり広い時間範囲に分布していることが分かる。最大値の枠で見ると14ジョブにおいてDFが109ミリ秒であるのに対し、PDFが108ミリ秒、PDFALLが107ミリ秒と大きな効果が出ていることが分かる。その他のジョブでもPDFおよびPDFALLの最大値の枠がDFよりも1または2個左に移っていて、PDFとPDFALLを比べてもほとんどがPDFALLの方がよい。また、最小値の枠は最大値ほど効果が出ていないが、その頻度はPDFやPDFALLの方がDFよりも良い。PDFとPDFALLを比べるとPDFの方が多少良いが、その差はそれ程大きくない。逆に、PDFALLの5ジョブで最小値の頻度が非常に多くなっているところもある。

4. 3問題ごとの平均スピードアップ

各機械数と各ジョブ数ごとの50問の平均スピードアップを図11、図12に示す。各スピードアップの値には

$$\text{スピードアップ} = \frac{\text{DFの探索時間合計}}{\text{PDFALLの探索時間合計}}$$

という計算式を用いている。

全体的に見て、機械数の変化によるスピードアップの変化はあまり見られない。しかし、11ジョブ以上では、多少機械数が少ないほうがスピードアップが大きい。

ジョブ数の変化で見ると、7ジョブまでのDFの実行時間の最大値が10秒程度のところでは、それ程大きなスピードアップは得られていない。またここではPDFALLの方が多少良いが差もそれほどではない。しかし、8ジョブより大きなところではPDFとPDFALLの差は大きくなり、10ジョブ以上でスーパー・リニア・スピードアップが見られるなど、明らかに全探索の効果が見られる。

4. 4全体的評価

前述の評価の仕方は問題別の特徴をつかむ上で意味はあるが、スピードアップの値が大きく異なるものどうして平均を取るにより不明瞭な部分も出てきてしまう。そこで、PDFとPDFALLで測定した各5000問全てを、それぞれ減速、リニア・スピードアップ、スーパー・リニア・スピードアップの3種類に分けて評価する。図13および図14はそれぞれPDFとPDFALLの結果を3種類に分類してそのスピードアップを示したものである。PDFでは減速が167例あったもののPDFALLではわずか68例にまで減っている。同様にリニア・スピードアップも減っており、増加分は全てスーパー・リニア・スピードアップとなっている。また、PDFALLの各スピードアップの値も減速以外は良くなっている。まとめるとPDFALLは、減速が全体の1.3%とかなり少なく、スーパー・リニア・スピードアップも全体のほぼ1割に達し、全体としてプロセッサの台数倍に近い31倍とPDFに比べて大きな効果が得られた。

5むすび

本報告では、nジョブ、m機械のフロージョブスケジューリングの総所要時間最小化問題を取りあげ、この問題における分枝限定法を並列化

実行時間（ミリ秒）

	101	102	103	104	105	106	107	108	109
5	20	470	10						
6	2	213	285						
7		54	371	75					
8		25	214	254	7				
9		17	81	294	107	1			
10		3	44	175	245	33			
11		6	23	100	235	130	6		
12		3	17	59	147	224	48	2	
13		5	15	37	90	198	134	20	1
14		2	16	26	63	125	180	74	14

最大：97時間47分12秒

図8 ジョブの変化による実行時間頻度分布（DF）

実行時間（ミリ秒）

	101	102	103	104	105	106	107	108	109
5		500							
6		500							
7		381	119						
8		163	337						
9		56	397	47					
10		22	261	209	8				
11		7	150	287	56				
12		5	90	222	162	21			
13		5	65	135	200	88	7		
14			57	86	148	166	42	1	

最大：3時間40分48秒

図9 ジョブの変化による実行時間頻度分布（PDF）

実行時間（ミリ秒）

	101	102	103	104	105	106	107	108	109
5	132	368							
6		500							
7		351	149						
8		163	337						
9		50	427	23					
10		22	283	192	3				
11		9	165	282	44				
12		7	90	232	155	16			
13		6	68	137	204	81	4		
14			61	87	151	160	41		

最大：2時間16分40秒

図10 ジョブの変化による実行時間頻度分布（PDFALL）

し、さらに全探索を導入して高速化した。これをn CUBE2上で実現しアルゴリズムの有効性を確認した。

現在、台数効果を確認するためにプロセッサ数が32台より少ない場合や問題データの条件を変えた測定を行っている。

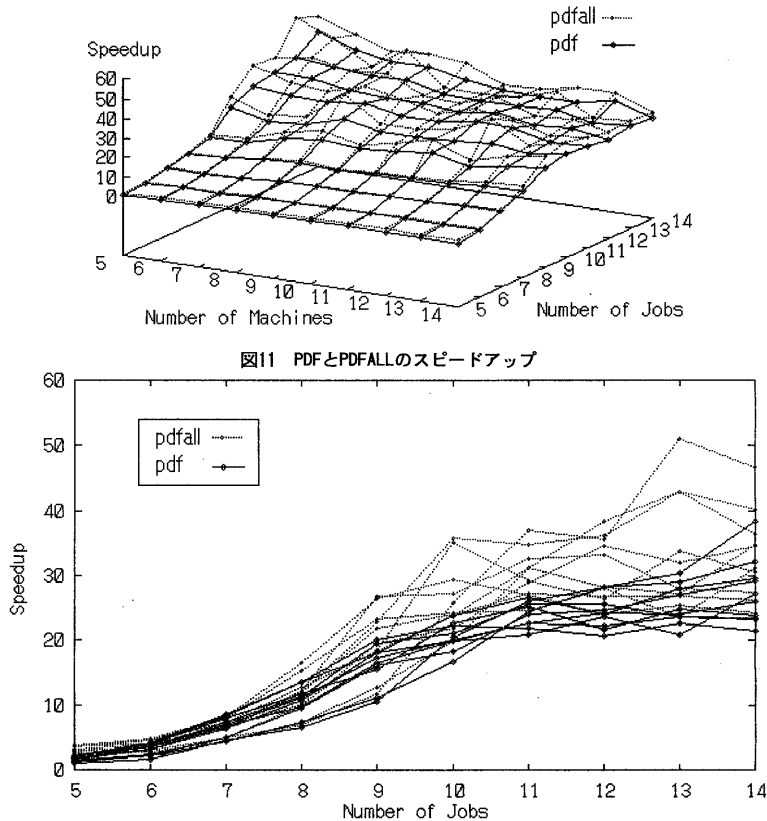


図11 PDFとPDFALLのスピードアップ

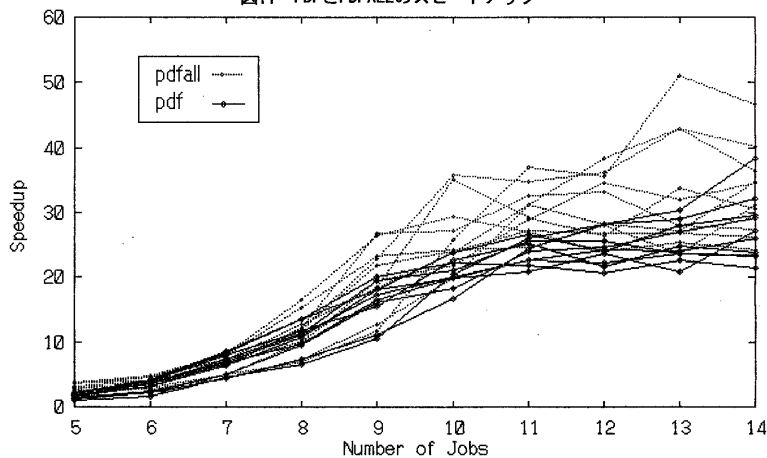


図12 ジョブの変化によるスピードアップ

case	count		DF	PDF
No		合計(秒)	4.4	5.3
Speedup	167	平均(秒)	0.026	0.031
		Speedup	1.0	0.8
Linear		合計(秒)	5759839.8	241935.8
Speedup	4631	平均(秒)	1243.7	52.2
		Speedup	1.0	23.8
Super		合計(秒)	1127626.9	27811.8
Linear	202	平均(秒)	5582.3	137.7
Speedup		Speedup	1.0	40.5
total	5000	合計(秒)	6887471.1	269752.8
		平均(秒)	1377.5	54.0
		Speedup	1.0	25.5

図13 分類別探索時間とスピードアップ (PDF)

case	count		DF	PDFALL
No		合計(秒)	2.1	2.6
Speedup	68	平均(秒)	0.031	0.038
		Speedup	1.0	0.8
Linear		合計(秒)	3368156.9	139817.4
Speedup	4443	平均(秒)	758.1	31.5
		Speedup	1.0	24.1
Super		合計(秒)	3519312.1	82569.1
Linear	489	平均(秒)	7196.957	168.9
Speedup		Speedup	1.0	42.6
total	5000	合計(秒)	6887471.1	222389.1
		平均(秒)	1377.5	44.5
		Speedup	1.0	31.0

図14 分類別探索時間とスピードアップ (PDFALL)

参考文献

¹ Kenneth R. Baker : "Introduction to Sequencing and Scheduling", John Wiley & Sons, Inc. (1974).

² Ignall, E., and Schrage, L. E. : "Application of the Branch and Bound Technique to Some Flow-Shop Scheduling Problems", Operations Research, Vol. 13, No. 3(1965).

³ 笠原博徳, 伊藤敦, 田中久充, 伊藤敬介: "実行時間最小マルチプロセッサスケジューリング問題に対する並列最適化アルゴリズム", 信学誌, vol. J74-D-1 No. 11 pp. 755-764 (1991).