システム評価 4−6
(2002. 8. 21)

マルチプロセッサスケジューラ HMSS の標準タスクグラフセット STG を用いた性能評価

益川 正如† 田頭 茂明† 藤田 聡†

† 広島大学 大学院工学研究科 情報工学専攻
〒739-8527 東広島市鏡山一丁目 4-1
{mawama,shigeaki,fujita}@se.hiroshima-u.ac.jp

あらまし: 我々が現在開発中のマルチプロセッサスケジューラ HMSS の性能を STG(標準タスクグラフセット) を用いて評価したので報告する。HMSS は分枝限定法に基づく最適化スケジューラである。分枝限定法の性能を大きく左右する要因のひとつに部分解の下界の与え方があるが、以下では、HMSS で採用されている精度の高い下界値の高速計算手法の効果について、他の下界との比較を通して詳細に検討する。実験の結果、全体の約 90%のインスタンスを 10 秒以内で解くことができ、その中には従来手法では 1 時間実行しても解が得られないものも 2 個含まれていた。

An Evaluation of Multiprocessor Scheduler HMSS using Standard Task Graph Set STG

Masayuki Masukawa† Shigeaki Tagashira† Satoshi Fujita†

† Graduate School of Engineering, Hiroshima University
Kagamiyama 1 chome 4-1, Higashi-Hiroshima, 739-8527 Japan

Abstract: This paper reports the result of performance evaluation of a multiprocessor scheduler HMSS by using the standard task graph set STG. HMSS is an optimization scheduler based on the branch-and-bound method, and adopts a novel technique to calculate a sharp lower bound in an efficient manner. The result of experiments implies that 90 % of the instances could be solved within 10 seconds, which contains two instances that could not be solved within one hour under conventional methods.

1 はじめに

本稿では、現在我々が開発中のマルチプロセッサスケジューラ HMSS (Hiroshima university Multiprocessor Scheduling problem Solver) の概要と、その性能を評価するためにおこなった一連の計算機実験の結果について報告する。HMSS は分枝限定法に基づく最適化スケジューラであり、(十分な時間さえかければ) 必ず最適解を出力することが保障されている。一般に分枝限定法による探索を効率よくおこなうためには、よりよい上界値をできるだけ早いタイミングで求めることと、各部分解に対して計算される下界値の精度をできるだけ向上させることが必要である (分枝限定法の詳細については後述する)。HMSS では上界を効率よく求めるための発見的手法として、笠原と成田によって提案された CP/MISF[2] を用いている[1]。また精度の高い下界を効率よく求めるための方法として、文献 [4] において我々が提案した下界値計算の高速化手法が用いられている。文献 [4] でも示されている通り、この高速化手法を用いることによって、これまでに用いられてきた下界値よりも精度の高い下界値を、オーダー的にそれらとほぼ同程度の計算時間で求めることが可能となる。

[1]CP/MISF はクリティカルパス法という一種のリストスケジューリング法に基づいており、実用的にも高い性能を持つことが知られている。

本稿の目的は、HMSS のマルチプロセッサスケジューラとしての性能を評価することである。具体的には、実データに基づく適切なベンチマークセットを用いて、HMSS がどの程度の時間で最適解を求めることができるのかを測定する。HMSS で採用されている下界値 (およびその計算手法) は、Fernandez と Bussell によって文献 [1] で提案された方法の改良版である。本稿ではこの下界を FBB と呼ぶことにする。実験では HMSS の性能のよさを示すため、比較対象として、Fernandez と Bussell によって提案された下界値 (これを下界 FB と呼ぶ)、および Fernandez によって示された Hu の下界の改良版 (これを下界 HF と呼ぶ) をそれぞれ HMSS 内に実装した。またベンチマークセットとしては笠原らによって開発された STG(標準タスクグラフセット) を用いた [3]。STG に含まれる 120 個のインスタンスに対しておこなった実験の結果、全体の約 90%のインスタンスを 10 秒以内で解くことができ、その中には HF や FB では 1 時間実行しても解が得られないものも 2 個含まれていた。

本稿の構成は以下の通りである。2 節ではモデルの定義と問題の定式化をおこなう。3 節では HMSS について説明し、4 節ではそこで用いられている下界について述べる。5 節では実験の結果を述べる。最後に 6 節では今後の課題について述べる。

2 モデル

タスク間の依存関係を示した非巡回有向タスクグラフ $G = (V, E)$ を入力として考える。ここで G の各ノード v は並列化された処理単位であるタスクをあらわし、有向枝 $(v, w) \in E$ は "タスク v の実行が完了するまで w の実行が開始されない" という先行制約をあらわす。有向枝 (v, w) に対して、v を w の先行ノード、w を v の後続ノードとそれぞれ呼ぶ。また各ノード v にはプロセッサ上での処理時間に対応する時間 $w(v)$ が割り当てられているものとする。先行ノードをもたないノードを入口ノード (entry node)、後続ノードをもたないノードを出口ノード (exit node) とそれぞれ呼ぶ。以下では、G はただひとつの入口ノード s とただひとつの出口ノード t をもつと仮定する。本稿では、タスクグラフ G で表現されるプログラムを同一の処理能力をもつ m (≥ 1) 台のプロセッサ $p_1, p_2, \ldots, p_m$ 上で並列に実行することを考える。ただし、タスク間のデータ転送や同期に要するコストは無視できる程度に小さいものとする。

以上のようなモデルのもとで、本稿で対象とするマルチプロセッサスケジューリング問題 (MSP) は次のように定式化される: 与えられたタスクグラフ G 上の $|V|$ 個のノードを、先行制約を満たし、かつ並列実行時間 (メイクスパン) が最小となるように m 台のプロセッサ上に割り当てよ。ただし入口ノード s の実行は時刻 0 に開始されるものとし、各プロセッサは任意の時刻で高々 1 つのタスクをノンプリエンプティブに実行するものとする。

3 HMSS

本節ではマルチプロセッサスケジューラ HMSS の概要について説明する。先にも述べた通り、HMSS では分枝限定法が用いられている。分枝限定法の基本アイデアは、与えられた問題をいくつかの部分問題に再帰的に分解し (分枝操作)、その中で最適解を含まないものを適宜カットする (限定操作) ことによって、実行時間の短縮を図ることである。限定操作は一般に、各部分解ごとに計算される下界値を用いておこなう。ここで部分解の下界値とは、その部分解を展開して得られるどの解も、その下界値よりも小さな値をもち得ないことが理論的に保障されている値のことをいう。部分解のカットは、その部分解の下界値がそれまでに得られているもっともよい暫定解 (上界値) と等しいか大きい場合におこなわれる。

HMSS で用いられる探索木は以下のように生成される。まず次のような性質を満たすタスクの部分集合 U $(\subseteq V)$ を考える: 「$u \in U$ ならば u のすべての先行ノード v に対して $v \in U$ である。」U によって導びかれる G の誘導部分グラフに対するスケジュールのことを、本稿では U に対する部分解と呼ぶ。HMSS で用いられる探索木の各節点は、それぞれある部分解に対応している。特に木の根は $U = \{s\}$ に対する部分解に対応しており、各節点 u からの子節点 v の生成は、u において割り当て可能なタスク (先行タスクが全て割り当てられている未割り当てタスク) の内の 1 つを、先行制約を満たす範囲でできるだけ早い時刻に割り当てることによっておこなうことができる。探索木のより詳細なデータ構造については [4] を参照されたい。

次に HMSS における探索木の展開方法について説明する。すでに述べた通り、HMSS では上界値を効率よく求める方法として CP/MISF[2] を用いている。

CP/MISF では探索木のある節点から生成された子節点たちは、以下の順番で (深さ優先的に) 展開される: 子節点で新たに部分解に追加されたタスクたちを比較し、1) 出口ノードまでの最長距離が長いほうを優先的に展開する、2) もし最長距離が同じ場合は、そのタスクの直接の後続タスク数が大きいほうを優先する、3) さらにその値も同じであればそれらからランダムに選択する。

4 下界

分枝限定法で用いられる下界に対して求められる主な要件は、精度の高さと計算量の少なさである。しかし一般にはそれらの間にはトレードオフの関係があるため、より短い計算時間で精度の高い下界を求めるための手法の開発が必要である。本節ではまず Fernandez と Bussell によって提案された方法 (下界 FB)[1] の概要を述べた後、我々が [4] で提案した新しい手法 (下界 FBB) について述べる。論文 [4] では各下界の効率のよい計算方法についても提案されており、オーダー的には下界 HF と同程度の時間で計算できることが示されているが、実際にプログラムを走らせてみると、細かいところで改良すべき点がいくつかみつかった。そこで以下では、現在我々が実装している HMSS 上でなされている高速化のための工夫のいくつかについてもあわせて紹介する。

4.1 下界 FB

与えられたタスクグラフ上の最長パスの長さを t_{cp} と記す。定義よりあきらかに、t_{cp} はメイクスパンの自明な下界値になっている。

すべてのタスクの実行が時刻 0 から時刻 T ($\geq t_{cp}$) までの間におこなわれるような状況を考える。区間 $[0,T]$ の任意の部分区間 $[\theta_1,\theta_2]$ を考え、(任意のスケジュールにおいて) その区間内で実行されなくてはならない仕事量の下界を $R(\theta_1,\theta_2,T)$ と定義する[2]。いま我々に与えられているプロセッサ数は m なので、各プロセッサは区間 $[\theta_1,\theta_2]$ の間に $R(\theta_1,\theta_2,T)/m$ 分の仕事を処理しなければならないが、もしこの仕事量が区間の長さ $(\theta_2-\theta_1)$ よりも大きければ、すべてのタスクの実行を時刻 T までに完了させることはできなくなる。このとき、実際の終了時刻の T からの増分の下界値は、$R(\theta_1,\theta_2,T)/m$ から $(\theta_2-\theta_1)$ を引いた値となる。以上の議論はすべての部分区間に対して成立するので、メイクスパンの自明な下界値 t_{cp} からの増分 q の下界値は以下のように与えられる

$$q \geq \max_{0\leq\theta_1<\theta_2\leq t_{cp}} \left\{ -(\theta_2-\theta_1) + \frac{R(\theta_1,\theta_2,t_{cp})}{m} \right\}$$

この値を用いてメイクスパンの下界 FB は $t_{cp}+\lceil q \rceil$ のように求められる。なお下界 HF は、上記の式において、θ_1 を 0 に固定するか、θ_2 を T に固定して得られた値である。

[2] $R(\theta_1,\theta_2,T)$ の具体的な計算方法については [4] を参照されたい。

4.2 下界 FBB

下界 FBB の基本アイデアは、「各区間 $[\theta_1,\theta_2]$ に対して必要な仕事を完了するために必要なプロセッサ数の下界 $\lceil R(\theta_1,\theta_2,T)/(\theta_1-\theta_2) \rceil$ を求める」という操作をいくつかの T について繰り返しおこなうことによって、上記の下界が与えられたプロセッサ数 m 以下となるような最小の完了時刻 T を求めることである。いま $m_L(T)$ を以下のように定義する:

$$m_L(T) = \max_{0\leq\theta_1<\theta_2\leq T} \left\lceil \frac{R(\theta_1,\theta_2,T)}{\theta_2-\theta_1} \right\rceil .$$

定義より、$m_L(T)$ はすべてのタスクの実行を時刻 T までに終了するために最低限必要なプロセッサ数である。もし $m \geq m_L(t_{cp})$ ならばメイクスパンの下界は t_{cp} であり、この方法ではこれ以上改良できない。しかし $m < m_L(t_{cp})$ の場合は、T の値を初期値 t_{cp} から少しずつ変化させ、$m_L(T) \leq m$ となるような最小の T を求めることで下界を改良することができる。このようにして得られた T の最小値がメイクスパンの下界 FBB である。

探索には二分探索法が段階的に用いられる。T と t_{cp} の差分をあらわす変数を Δ とする。Δ の値は 1 に初期化され、$m \geq m_L(t_{cp}+\Delta)$ をはじめて満たすまで $\Delta := 2*\Delta$ を繰り返す。その後で、区間 $[t_{cp}+\Delta/2, t_{cp}+\Delta]$ の中を二分探索して $m \geq m_L(t_{cp}+\Delta')$ を満たす最小の Δ' を見つけていく。この方法により、最終的に求められる差分値 Δ' に対して $2\lceil \log_2 \Delta' \rceil$ 回の検査で求めるべき下界値を得ることができる。

4.3 下界の計算時間の高速化

文献 [4] で我々が提案した手法は、オーダー的には HF と同程度の実行時間を実現するものであったが、実際の実行時間には 100 倍程度のひらきがあった。

以下では、下界FBBの計算時間を抑えるためにおこなったいくつかの改良点を述べる(最初の2点については、FBBだけでなく他の下界についても同様に適用されることに注意されたい)。

- 下界値FB、FBBの計算では、$0 \leq \theta_1 < \theta_2 \leq T$であるようなすべての$\theta_1, \theta_2$に対して関数$R$の値を計算することが求められていた。しかし(根節点以外の)各部分解では、すでにいくつかのタスクがプロセッサに割り当てられており、それらの時刻を、各プロセッサにタスクが割り当てられているような時刻にまでさかのぼって設定する必要はない。すなわち、考慮すべきθ_1, θ_2の組合せの種類をこれから割り当てがなされる時刻の範囲に限定することができる。このような改良は、探索木の葉に近い部分で特に効果的であると考えられる。
- 各節点の下界値は、少なくともその親節点の下界値以上であるから、その節点の下界値の初期値を親節点の下界値に設定しておき、具体的な下界の計算はその値で枝刈りができなかったときにのみおこなうように変更する。
- 下界FBBの計算(二分探索)において、Δの初期値を"親の下界値 $- t_{cp}$"とし、その後、$m \geq m_L(t_{cp} + \Delta)$をはじめて満たすまで$\Delta$の値を1ずつ増やすように改良する(実際には増分の値は1か2であることが多い)。

5 実験

本節では提案手法の有効性を調べるためにおこなった実験の結果を示す。実験環境は以下の通りである。CPU: Pentium4 Xeon (1700MHz), Memory: 1024MB, OS: FreeBSD4.4。実験では、3種類の下界(FBB, FB, HF)のそれぞれについて、探索木の各レベルでおこなわれた枝刈り数、解法全体の総実行時間、および下界の平均計算時間を測定した。また実験には笠原らによって開発された標準タスクグラフセット(STG)(正確にはプロトタイプ版)を用いた[3]。

現在公開されているSTGはタスク数が1000未満のインスタンス120個とタスク数が1000以上のインスタンス180個から成っており、今回の実験ではタスク数が1000未満のインスタンスに限定して考察をおこなうこととした。実験の結果、これらの120個のインスタンスは、以下のような3つのタイプに分類されることがわかった。

タイプA	最適解とクリティカルパスの長さt_{cp}が一致するもの	53個
タイプB	最適解はt_{cp}より大きいが、CP/MISFによって最初に得られる上界値と木の根節点の下界値とが一致するもの(実験で評価した120個のインスタンスすべてにおいて、根の下界値は、HF、FB、FBBのいずれでも同じ値になった)。	33個
タイプC	それ以外	34個

タイプAでは、簡単な下界で解を求めることができる。また、タイプBでは、根節点から最初に発見される上界値を与える葉節点とを結ぶパスのみが生成され、そこからの枝分かれはおこらない。下界値の計算は根節点で一度だけおこなわれ、それ以外の節点の下界値は初期値(親節点の下界値)がそのまま使われることになる。したがってタイプBのインスタンスは、(HFでも)ほぼ線形時間で解くことができる。このことから以下では、タイプCのインスタンスについてさらに詳細に調べていくことにする。

タイプCの34個のインスタンスのうち、1時間以内に解くことのできた18個のインスタンスに対するHMSSの実行時間と各下界の平均計算時間をそれぞれ表1、表2に示す。ここで*はタイムオーバーをあらわし、1時間の計算では解けなかったことを示している。表1より、3つの例外(012,014,018)を除いて、FBBを用いた場合は10秒以内に答えが求まっていることがわかる。それらの例外のうちの2つ(014と018)はHFやFBを用いた場合には1時間で答えが求まらなかったインスタンスであり、このことから、FBBが十分高速であること、あるいはFBBには十分な高速化の効果があることがわかる。前述の3つのインスタンスを除いた15個のインスタンスのうち、総実行時間でHFを改良しているものが11個、FBを改良しているものが5個ある。また改良できていないものについても、その実行時間の差は5秒以内であり、実用的な観点からはほぼ同程度の性能であると言える。

FBBが他の手法に比べて長い実行時間を要する場合があるのは、ひとつの部分解に対して下界を計算する時間が他の手法に比べて長いためである。その

表 1: タイプ C の各インスタンスに対する HMSS の実行時間 (*はタイムオーバーを表す)

ID	タスク数	総実行時間 [sec]		
		FBB	FB	HF
008	399	5.26	78.10	575.00
012	939	12.90	0.77	0.67
013	799	2.63	*	*
014	636	375.00	*	*
018	730	67.50	*	*
019	617	2.08	1.48	8.34
059	421	1.51	1.56	5.42
063	713	2.66	0.99	*
111	532	2.12	0.29	0.43
155	406	1.81	0.70	1.48
163	900	4.30	0.39	0.37
165	560	0.34	*	*
166	745	3.52	3.15	105.00
205	235	2.65	0.17	0.55
209	464	1.09	0.16	10.50
212	702	3.93	3.20	24.80
214	607	2.42	224.00	1064.00
254	250	6.80	6.55	64.60

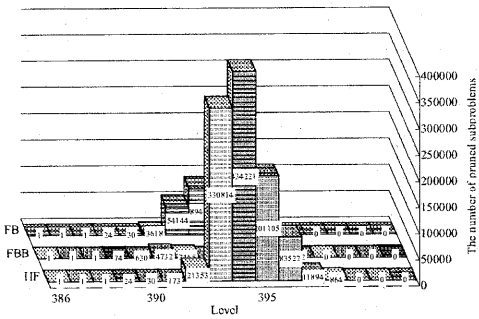
表 2: 下界の 1 回あたりの平均計算時間 (*はタイムオーバーを表す)

ID	タスク数	下界の計算時間 [msec]		
		FBB	FB	HF
008	399	0.55	0.42	0.41
012	939	846.00	15.40	10.50
013	799	56.50	*	*
014	636	0.95	*	*
018	730	0.78	*	*
019	617	102.00	0.74	0.71
059	421	62.50	0.53	0.50
063	713	72.10	1.17	*
111	532	25.60	0.98	0.74
155	406	64.60	0.54	0.48
163	900	241.00	11.00	7.70
165	560	2.08	*	*
166	745	124.00	1.33	1.24
205	235	6.29	0.28	0.24
209	464	21.50	1.39	0.52
212	702	115.00	1.17	1.12
214	607	25.90	0.74	0.74
254	250	20.80	1.94	1.60

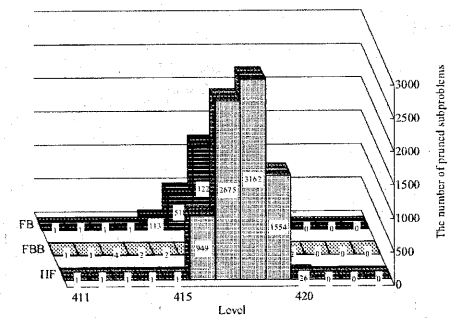
具体的な値を表 2 に示す。この表から、たとえばインスタンス 019 に対しては、HF と FB ではそれぞれ 0.71 msec、0.74 msec で部分解ひとつあたりの下界の計算がおこなわれていたのに対し、FBB における下界の計算は部分解ひとつあたり 102 msec かかっていることがわかる。これらの計算時間のうち、根節点の下界値 (探索のいちばん最初に計算される) の計算と、それ以外の節点の下界値 (親節点からコピーされた初期値をつかって枝刈りがおこなえなかったときに計算される) の計算とに、それぞれどのくらいの時間がかかったのかをさらに詳細に調べてみたところ、14 個中 11 個のインスタンスにおいて[3]、根節点での計算時間の割合が 90%を越えていることがわかった。試みに根節点以外での下界の計算に要した時間を FBB と FB で比較してみたところ、それらの間にはほとんど差がなかった。すなわちこれらのインスタンスでは、根節点以外での下界の計算において、4.3 節で述べた高速化の効果が表れていることがわかる。また根節点での計算時間の割合が低いインスタンスに関しては (たとえばインスタンス 008 では、この割合は 20%ほどであった)、高速化の効果がそのまま下界の平均計算時間にあらわれていることも表から見て取れる。

一方、FBB によって高速化が達成されているインスタンスでは、下界値の改良による枝刈りが効果的におこなわれていると予想される。この高速化の度合いは、いちばん大きなもので 10000 倍にもなる (たとえばインスタンス 165)。下界値の改良が枝刈りの効率化にどのような影響を与えているのかを調べるため、枝刈りの数を探索木の各レベルごとに評価してみた。図 1 に結果の一部を示す。ここで図 (a) と (b) はそれぞれインスタンス 008 とインスタンス 059 に対する結果をあらわしている。図の横軸はレベルであり、各レベルごとの枝刈り数がヒストグラムの形で表示してある (タスク数に入口ノードと出口ノードの数を加えたものが最大のレベルになっているこ

[3]この評価では HF でタイムアウトになったインスタンスは除外している。



(a) インスタンス 008



(b) インスタンス 059

図 1: 探索木の各レベルごとの枝刈り数.

とに注意されたい)。図からあきらかなように、FBB は FB や HF よりもより高いレベルでの枝刈りが多くなされていることがわかる。実際、インスタンス 008 では、レベル 389 での FBB の枝刈り回数が 74 回であるのに対し、FB と HF ではそれぞれ 24 回しかおこなわれていない。またインスタンス 059 では、レベル 413 での枝刈りが FBB では 4 回おこなわれているのに対し、FB と HF の枝刈り回数はそれぞれ 1 回で、HF や FB では無駄な展開がより多くおこなわれていることがわかる。

6 おわりに

本稿では、STG ベンチマークセットを用いてマルチプロセッサスケジューラ HMSS の性能を評価した。実験の結果、評価したタイプ C のインスタンスの約 40 % で提案手法による高速化の効果が見られ、さらに全体の約 90% を 10 秒以内に解く事ができた。

STG のタスク数 1000 未満の 120 個のインスタンスのうち、我々の手法で 1 時間以内に答えが求まっていないものがまだ 16 個残っている。今後の課題として、処理をさらに高速化することで、それらのすべてを実用的な範囲の時間内で解くことがあげられる。そのための具体的な方法として、下界 FBB のいっそうの改良と、並列処理の導入を考えている。

謝辞： 本研究の一部は、日本学術振興会科学研究費補助金一般研究 (C)(2)(課題番号 13680417) および同補助金奨励研究 (A)(課題番号 13780335) による。

参考文献

[1] E. B. Fernandez and B. Bussell. "Bounds on the number of processors and time for multiprocessor optimal schedules," *IEEE Trans. Comput.*, C-22(8):745–751, Aug. 1973.

[2] H. Kasahara and S. Narita. "Practical multiprocessor scheduling algorithms for efficient parallel processing," *IEEE Trans. Comput.*, C-33(11):1023–1029, Nov. 1985.

[3] `http://www.kasahara.elec.waseda.ac.jp/schedule/index.html`

[4] 藤田 聡, 益川 正如, 田頭 茂明. マルチプロセッサスケジューリング問題に対する分枝限定解法の下界の改良に基づく高速化について, 並列処理シンポジウム (JSPP2002), pp.289–296, May 2002.