成蹊大学理工学研究報告
J. Fac. Sci.Tech., Seikei Univ.
Vol.50 No.1 (2013)pp.17-28

通信遅延を含むタスクスケジューリング問題の並列分枝限定法に基づく探索解法

栗田　浩一*1，甲斐　宗徳*2

A Parallelized Branch-and-Bound Search Method for Task Scheduling Problem with Consideration of Communication Overhead

Kohichi KURITA*1, Munenori KAI*2

ABSTRACT：Since the task scheduling problems in the multiprocessor environments belong to the class of strong NP hard combinatorial optimization problem, the depth first search algorithms based on branch and bound(B&B) method are most effective to find an optimal solution. In order to reduce the search time with B&B method, it is the most important key to construct search algorithms with the way to bound many branches in the search space by more accurate lower bounds. However, it seems that there are no algorithms to create such lower bounds with consideration of processing system environments. In this paper, we propose three algorithms to create the lower bounds with consideration of number of processors, the lower bounds with consideration of processing time of successive tasks, and the lower bounds re-calculated in search process. Our experiments show that these algorithms give more accurate lower bounds and improve efficiency of search.

Keywords：Task scheduling, Combinatorial optimization problem, Branch and bound

(Received March 31, 2013)

1．はじめに

タスクスケジューリングは，並列処理環境で各マシンに最適なタスクの割当てを決定し処理パフォーマンスを最大限に引き出す技術である。現在までにグリッドコンピューティングやクラスタ計算機の普及から，タスク間の依存関係により生じるノード／プロセッサ間の通信遅延を考慮したスケジューリング手法の需要が高まっている。

このタスクスケジューリングは，強NP困難な組合せ最適化問題に属する[1]。従って最適解を得るには本質的に分枝限定法による探索解法が必要になる。筆者は，通信遅延を考慮しないタスクスケジューリング問題で既に有効性が証明されている逐次最適化解法DF/IHS(Depth First/Implicit Heuristic Search)法に通信遅延の組合せを足し合わせ，それを複数コアで並列処理する並列最適化解法の研究を行っている[2]。DF/IHS法は，分枝限定法による解法を行うため，分枝操作に各タスクのプライオリティを使用した部分スケジューリングと限定操作に下限値を使用した枝切りを行う。このプライオリティと下限値にはタスクの処理時間のみを考慮し，当該タスクからエンドノードまでに最低限必要とされる経路長(以下CP:Critical Path長)が使用されてきた。しかし，通信遅延の有無は，このCP長に変化を与えてしまう場合がある。従ってこのまま使用してもこの下限値に本来の性能を望むことはできない。昨年度までの本研究室の研究成果で，暫定的に通信遅延を算入したREDIC(Remaining Distance Including Communication Overhead)法が考案された[3]。この手法は下限値を求める際に該当するタスクから限定された範囲内のタスクを考慮に入れるものであった。そのため，まだ発展の余地がある。従って，本稿ではREDIC法を改良することで探索効率の向上を図った。

*1：理工学研究科理工学専攻博士前期学生
*2：理工学研究科理工学専攻教授 (kai@st.seikei.ac.jp)

2．問題定義

タスクとは，実行プログラムを並列処理するために分割された部分処理である。タスク集合はタスクをノード，依存関係を有向エッジで表し，入口と出口にダミーノードを設置したタスクグラフと呼ぶDAG(Directed Acyclic Graph)で表現することができる(図 1)。タスクiの処理に要する時間をw_iと表す。これを同一の処理能力を所持したm(>=1)台のプロセッサ(以下PE:ProcessorElement)に割当て，タスク毎に並列実行させる。さらに本研究では，直接先行後続関係にあるタスク同士が異なるPEに割当てられた場合，データ依存解決に必要な通信にともなう遅延時間を考慮する。そのため，エッジに通信遅延時間を付加する。この場合，通信遅延時間はタスクiとその親タスクのタスクjが互いに異なるPEに割当てられた場合，エッジに付加された通信遅延$C(i,\ j)$のコストを支払う。もし同一のPEに割当てられた場合は，通信遅延はゼロとなる。

図 1 を例にすると，タスク 4 の処理時間は$w_4 = 1$を支払い，タスク 4 とその親タスクのタスク 2 が互いに異なるPEに割当てられた場合，エッジに付加された通信遅延$C(4,\ 2) = 2$のコストを支払う。

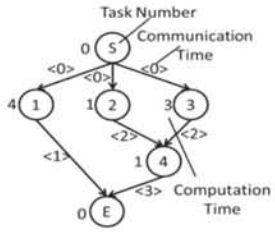

図１　サンプルタスクグラフ

3．スケジューリングアルゴリズム

3.1　逐次最適化解法DF/IHS法

厳密解を求めるための実用的な最適化アルゴリズムとしては，DF/IHS法が笠原らにより提案されている[7]。DF/IHS法は，分枝限定法による逐次最適化アルゴリズムである。

図 2 は，図 1 のタスクグラフで 2PEに割り当てるスケジューリング問題を実際にDF/IHS法で探索した場合の探索空間である。RT(Ready Task)はその時点でアイドルPEに割り当て可能なタスクの集合を表す。TS(Task Selection)は，RTから実際にアイドルPEに割当てたタスクの集合を示す。こちらは，RTからCP/MISF(CP/Most Immediate Successors First)法[7]のヒューリスティック的に優先順位の高いタスクの順序で割当てる。Idle Taskの割り当ては，PEをアイドル状態にすることを意味する。DF/IHS法は図2のような探索空間を左端から右端にかけて深さ優先探索を行う。

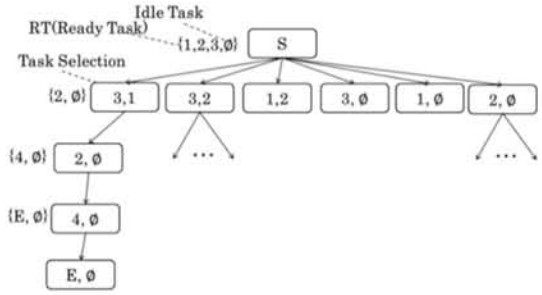

図 2 DF/IHS法の探索空間

筆者は，この手法に割当てPEの選択と通信の送受信の組合せを加えることで，通信遅延を考慮している。これは組合せ最適化問題の計算複雑度を増やし，最適解の発見をより困難にする。

DF/IHS法はCP/MISF法より，各タスクからエンドノードまでのCP長を算出し，その値を下限値とすることで枝切りが効率的に行われる。しかし，通信遅延の有無は，この経路長を伸ばしてしまう場合がある。そのため，CP/MISF法によって求められた下限値では，限定操作において過小評価されてしまう可能性がある。従って，この下限値の精度を向上させることで枝切り効果を上げ，探索効率の向上が期待できる。

4．下限値

4.1　スケジューリング手法における下限値

スケジューリング手法における下限値は，スケジューリングを行う問題対象にとって最低限必要となるメイクスパンを表す。これは，問題に対する有用なメイクスパンの目安と，同時に最適化解法を行う際の最適解の目安にもなる。本研究室では，この下限値にREDIC法[3]の値が使用されてきた。

4.2　REDIC法

REDIC法は下記の条件下で算出する。

- 直接後続タスクの通信遅延のみを考慮
- 処理 PE 数は無制限

更に、計算を行う上で下記の定義を導入する。

- A:最早完了時間を決定するタスク
- P_A:タスク A を処理する PE
- $P_{\sim A}$:タスク A を処理しない PE
- E:タスク A の直接後続タスク群
- $P_A(E)$:P_A に割り当てられたタスク A の直接後続タスク群
- $P_{\sim A}(E)$:$P_{\sim A}$ に割り当てられたタスク A の直接後続タスク群
- e_{sA}:タスク A の最早開始時間(earliest Start time)

- e_{CA}: タスク A の最早完了時間(earliest Completion time)
- ct_A: タスク A の完了時間(Completion time)

e_{CA}は，e_{sA}にw_Aを足し合わせたものを示す。計算はエンドノードからスタートノードに向かって計算を行う。下記は下限値を求める当該タスクでの計算方法である。このとき，エンドノードのe_{sA}，e_{CA}は共に自身の処理時間とする。

① 直接後続タスクを最早開始時間の昇順でP_Aに割当てる。

$$P_AL = \max_{i \in P_A(E)}(ct_i), \quad ct_i = e_{ci} + \Delta,$$

$$\Delta = \begin{cases} ct_{i-1} - e_{si} & if(ct_{i-1} > e_{si})のとき \\ 0 \end{cases}$$

② $P_A(E)$の数が 1 であれば計算を終了。そうでなければ，最早完了時間に通信遅延を足し合わせたものが最小となるタスクを取り出し，$P_{\sim A}$に最早完了時間の昇順で割当てる。

$$P_{\sim A}L = \max_{i \in P_{\sim A}(E)}(ec_i + c(A,i))$$

③ P_ALと$P_{\sim A}L$を比較し$P_{\sim A}L$の方が大きければ計算を終了。そうでなければ①に戻る。

最終的に計算終了時のP_ALまたは$P_{\sim A}L$の大きい値がタスクAの最早開始時間を示し，w_Aを足した値が最早完了時間と下限値になる。

既にREDIC法による下限値の有効性は証明されている[3]。しかし，この下限値はPE数を考慮しない環境を条件にタスクの処理時間と互いに結ばれたエッジの通信遅延のみしか考慮していない。これは，分枝限定法による探索を行う際に任意の問題に対して十分な下限値の精度を所持できていない。従って，別の要素を追加することでこの下限値の精度向上を目指した。

4.3　プロセッサ数を考慮した下限値

PE数を考慮した下限値は現在までにいくつか考案されている[4],[5],[6]。現存の中で最も効果が高いとされているのが，Fernandezらによる下限値[6]である。これはPE数を限定しない環境で各タスクの活動範囲を算出し，CP長内の各インターバルにおけるタスクの重みを計算する。そこから，PE数が限定された環境においてインターバル内で処理しきれないタスクの重みを算出するものである。今回導出する下限値t_Lは下記の式から求められる。

$$t_L = t_{REDIC} + \lceil q \rceil$$

$$\mathrm{q} = \max_{[\theta_1,\theta_2]}\left[-(\theta_2 - \theta_1) + \frac{1}{m}\sum_{i \in N}\min\left[e_i(\theta_1,\theta_2),\ l_i(\theta_1,\theta_2)\right]\right]$$

$$\min\left[e_i(\theta_1,\theta_2),\ l_i(\theta_1,\theta_2)\right] = \min[(e_{ci} - \theta_1), w_i, (\theta_2 - \theta_1), (\theta_2 - l_{si})]$$

ただし，ここでt_{REDIC}はREDIC法により計算された下限値である。qはPE数が限定された環境においてインターバル内で処理しきれないタスクの重みである。CP長内の各インターバルはθ_1，θ_2で表す。更に，l_{si}はタスクiの最遅開始時間(latest start time)を表す。[6]が提供した下限値にはこの最早完了時間，最遅開始時間の各々に通信遅延が含まれない。従ってこれらに通信遅延を算入し，前述した式を適用することでPE数を考慮した下限値を生成した。

4.3.1　通信遅延を考慮した最早完了時間・最遅開始時間

通信遅延を考慮した最早完了時間と最遅開始時間はREDIC法から計算できる。最早完了時間は，REDIC法をスタートノードからエンドノードに掛けることで求まる。最遅開始時間は，REDIC法で求められた各タスクの下限値をスタートノードの下限値から引いた値が各タスクの最遅開始時間となる。下記に式を示す。

$$l_{si} = Re_s - Re_i$$

ここで，Re_sはスタートノードの下限値をREDIC法によって計算した値，Re_iはタスクiの下限値をREDIC法によって計算した値とする。

更に，この最早完了時間と最遅開始時間はタスクグラフからスタティックに求められる値とスケジューリング中に先行タスクのスケジュール状況から求められる値の 2 つがある。スケジューリング中に最早完了時間と最遅開始時間を更新し，そこから前述した式を適用することで，算出されたqからより積極的な枝切りが行える。しかし，前述した式は計算量$\mathrm{O}(\mathrm{V} * Re_s{}^2)$を費やすため，スケジューリング中にこの下限値を求める場合，この計算量が常に必要とされる。ここでVはタスクグラフのタスク数とする。この問題に対して下記の式を使用する。

$$\text{並列度para} = \frac{\text{タスクの処理時間の総和}}{\text{CP長}}[9]$$

並列度paraは任意のタスクグラフに対してCP長内で処理を終えるための必要なPE数の概算値となる。この値を閾値とし，並列度paraよりも小さい値を割当てPE数とするスケジューリング問題では探索中にPE数を考慮した下限値生成を行う。もしそうでなければ，探索中にPE数を考慮した下限値生成を行わない。

4.3.2　プロセッサ数を考慮した下限値の評価

PE数を考慮した下限値を用いてREDIC法による下限値との比較評価を行う。評価関数を以下に示す。

● スタートノードの下限値と最適解との近似率

評価を行うタスクグラフのパラメータは，タスク数:20，タスクの処理時間:1~30 の一様乱数，通信遅延時間:CCR=1.0，直接先行・後続タスク数 1~3 の一様乱数とした。ここでCCR(Communication to Computation Ratio)はグラフ上の通信遅延の総時間をタスクの総処理時間で除算した比となっている。このパラメータを所持したタスクグラフを 60 題用意して評価を行う。使用したマシンは，CPU : Intel(R) Xeon(R) X5690 @ 3.47GHz×2，OS:Linux，RAM : 50GB，Compiler : gcc 4.1.2 である。

4.3.3　評価結果

REDIC法とPE数を考慮した下限値とで評価を行う。PE数が関係してくる評価になるため，60 題のタスクグラフをPE数2~4 に割り当てる各々のスケジューリング問題を解き，平均値を取る。結果を以下に示す。

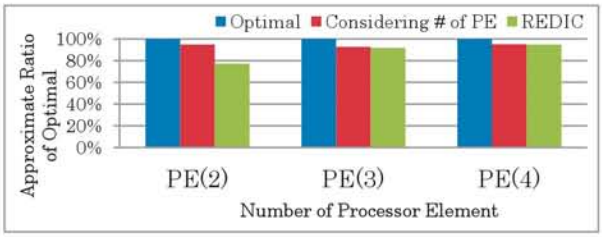

図 3　PE数を考慮した下限値生成の評価

図 3 は，最適解にどの程度各値が近似しているかを示す。図の横軸はPE数を表し，縦軸は最適解との近似率を表している。Optimalは最適解を表すため，近似率は常に100%となる。Considering # of PEはPE数を考慮した下限値となっている。

最も顕著な違いを示したのはPE数を2台とした場合である。REDICでは最適解との近似率に平均で 77.35%，最大で 95.7%となった。対してPE数を考慮した場合は，平均で 94.8%，最大で 98.1%と非常に最適解に近い下限値を生成することができた。PE数が 3，4 台の場合はどちらの下限値も顕著な違いは見られなかったが，下限値は最適解にどれだけ近似できるかでその有用性，有効性が劇的に異なるためREDICよりも近似した値を算出できただけでもメイクスパンの有用性，限定操作の有効性と共に効果があると考えられる。

4.4　後続タスクの処理時間を考慮した下限値計算

依存解決における通信遅延の値が多大である場合(図 4)，最適なスケジュールは通信の発生を抑えて，ある程度のPEにタスクをまとめた形(図 5)となる場合が多い。この場合，最適解とREDIC法によって求められた下限値に大きな差が生じてしまう。これは，REDIC法の計算目的が通信遅延を下限値に組み込むとしているためである。もし，図 5 の各タスクを逐次に実行するようなスケジュール長に下限値を近づける場合，当該タスクから全ての後続タスクのPEに対する割当てを考慮して求める必要がある。しかし，全ての後続タスクを下限値計算の要素に加えることは組合せの考慮から探索を必要とし，後々の最適解発見のことも考えると，探索の探索となってしまう。従って，今回は後続タスクが依存解決に通信を必要としない，直接先行タスクと同PEに割当てられる組合せを取る場合に後続タスクの処理時間をREDIC法の計算の要素に入れることで下限値の向上を目指した。

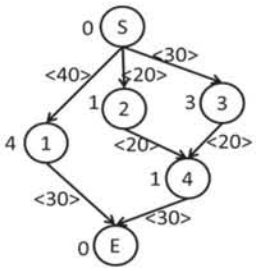

図 4　サンプルタスクグラフ

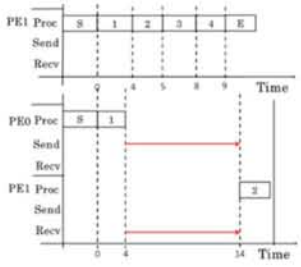

図 5　スケジュール例

4.4.1　計算方針

REDIC法の計算では，各タスクの下限値，最早開始時間，最早完了時間は図4を例にすると表1のようになる。

表 1　各タスクのREDIC法による値

Task	LB(LowerBound)	e_s	e_c
S	8	8	8
1	4	0	4
2	2	1	2
3	4	1	4
4	1	0	1
E	0	0	0

表 1 よりスタートノード以外のタスクはエンドノードまでの経路長に通信遅延は入れられないため，各値は正しいものとなっている。従って，スタートノードの下限値を向上させることで図 4 の逐次実行による最適解 9 に下限値を合わせる必要がある。

スタートノードの下限値をREDIC法によって計算する場合，計算に加わる要素はタスク 1，タスク 2，タスク 3 のみとなる。ここにタスク 4 を加えて計算してもREDIC法のアルゴリズムでは下限値として値を算出することができない。この問題に対して直接後続タスクの下限値，最早開始時間，最早完了時間を再帰的に更新しながら値を求める方法を提案する。

4.4.2　ダミーエッジの挿入

本来，スタートノードの下限値を最適解 9 と算出できない理由は，タスク 2，タスク 3 をREDIC法で計算する際にタスク 1 の要素を取り込めないためである。図 6 に例を示す。図 6 はエンドノードからスタートノードに向けて時刻 8 の間に各タスクが最早開始時間で存在する図を表している。図 6 より，タスク 2，タスク 3，タスク 4 は実際にPEに割当てを行うとタスク 1 の存在から表 1 の最早開始時間から実行することはできない。仮にタスク 2，タスク 3，タスク 4 を先に割当ててもタスク 1 が表 1 の最早開始時間から実行することはできない。タスク 1，4 の間でどちらを先に実行した方がいいのかはこの時点では決定することができない。しかし，タスク 1，2，3，の間で考えた場合，タスク 1 はタスク 2，3 よりも最早開始時間が早いため，タスク 1 を先に実行すればスタートノードの最早開始時間が冗長に延びることはない。従って，タスク 2，タスク 3 の下限値計算をするのにタスク 1 の要素を取り込んでおけば，結果としてスタートノードは全ての後続タスクの処理時間を考慮できる。

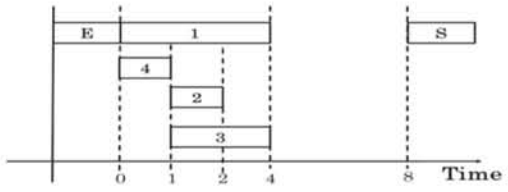

図 6　表 1 の最早開始時間の各タスクの存在図

タスク 2，タスク 3 の下限値計算にタスク 1 の要素を取り込むには疑似的な依存情報(ダミーエッジ)を持たせることで解決できる。図 5 に例を示す。 図 5 ではタスク 2 の直接後続タスクに疑似的にタスク 1 を加えて，タスク 2 の下限値をREDIC法で計算することで，タスク 1 の処理時間を計算の要素に加えることが出来る。更に，ダミーエッジの通信遅延時間を無限大とすれば，通信遅延を下限値の経路長に含むこともない。タスク 2 の下限値をREDIC法で再計算すると下限値は 6，最早開始時間は 5，最早完了時間は 6 となる。このとき，スタートノードの下限値をより正確にするためにタスク 2 とタスク 1 に依存関係を持たせている。本来このタスク 2 からエンドノードまでの経路にタスク 1 は存在しないため，スケジューリングを行う際のタスク 2 の下限値は表 1 の値 2 が正しい。

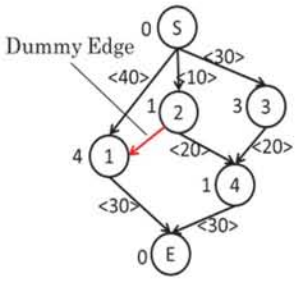

図 7　ダミーエッジの挿入

4.4.3　後続タスクの処理時間を考慮するアルゴリズム

ダミーエッジを設けることで，後続タスクの処理時間を下限値計算に組み込むアルゴリズムを下記に示す。

① P_A に割り当てられたタスクの中で、当該タスクよりも先に同 PE に割り当てられたタスクと先行依存の有無を判定する。有れば、次のタスクの判定に移る。全てのタスク判別を終えれば終了。

② 先行依存がなければダミーエッジを設け、エッジの通信遅延時間を∞に設定する。

③ ダミーエッジを設けた状態で当該タスクの下限値を再度 REDIC 法によって再計算する。

④ ③の再計算により当該タスクの最早開始時間、 最早完了時間を更新し①に戻る。

上記のアルゴリズムを用いて，実際に図 4 のグラフから下限値を求める。

まずREDIC法の計算アルゴリズムから当該タスクと直接後続タスクは同PEに割当てられる組合せを下限値として持つ。このグラフの場合，上記のアルゴリズムが適用されるのはスタートノードの下限値を計算するときである。

スタートノードの下限値はREDIC法のアルゴリズムから図 8 のスケジュールを持つ。

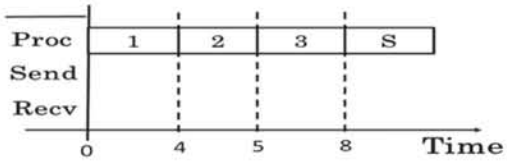

図 8　REDICによるスタートノードの組合せ

図 8 の場合，タスク 2 を当該タスクとした場合，当該タスクよりも先に同PEに割当てられているタスク 1 と依存関係を所持していないのでここに図 5 と同様のダミーエッジを設ける。ダミーエッジを設けた状態でタスク 2 の下限値を再計算する。再計算した結果を用いるとスケジュールは図 9 となる。

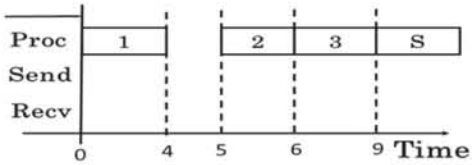

図 9　タスク 2 の下限値を再計算した結果

図 9 からスタートノードの下限値 9 を得ることが出来る。勿論，アルゴリズムの流れに従えば，次にタスク 3 とタスク 2 の間に依存関係があるかを判別する。この場

合ダミーエッジを設ける条件に入るのでタスク 2，タスク 3 の間にダミーエッジを設けタスク 3 の下限値を再計算する。結果としては，図 9 と同じスケジュールを得られる。

このアルゴリズムは直接後続タスクの下限値を再計算していく流れとなる。従って，当該タスクからエンドノードまでの後続タスク全ての下限値を再計算することが最も性能を良くするが，計算量が増えるため，トレードオフとなる。

4.4.4　後続タスクの処理時間を考慮した下限値の評価

後続タスクの処理時間の考慮による下限値を用いてREDIC法による下限値との性能差を示す。評価関数は以下とする。

● スタートノードの下限値と最適解との近似率

評価を行うタスクグラフのパラメータは，タスク数：10，タスク処理時間：1～20 の一様乱数，直接先行・後続タスク数：1～3 の一様乱数とした。このパラメータを所持したタスクグラフを 60 題用意した。実験環境は4.3.2項と同様である。

4.4.5　評価結果

後続タスクの処理時間の考慮による下限値とREDIC法によって求められた下限値とで最適解との近似率を，タスクグラフ 60 題から平均値を読み取って評価した。この評価は通信遅延時間の変動により，どの程度下限値が最適解に近似するかが重要になる。従って，CCR=1.0, 2.0, 3.0, 5.0, 10.0 と値を変えて評価を行った。結果を図 10 に示す。図 10 は，横軸をCCR値，縦軸を最適解との近似率としている。更に，後続タスクの処理時間を考慮したアルゴリズムは下限値を決定する当該タスクからエッジ何本先までの後続タスクを考慮するかで性能に変化ができるため，当該タスクからエッジ 2 本目までの後続タスクを考慮した場合をPath2，当該タスクからエッジ 3 本目までの後続タスクを考慮した場合をPath3，同様の意でPath4，Path5，Path6 までを評価した。従って，Path1 はREDIC法による計算と等価になる。

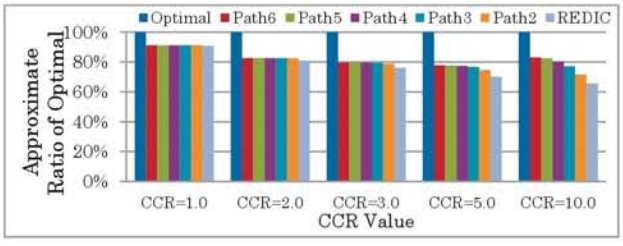

図 10　後続タスクの処理時間を考慮した下限値生成の評価

図 10 では，最適解にどの程度各下限値が近似しているのかを示す。そのため，最適解となるOptimalは近似率100%を示す。

タスクグラフの通信遅延時間をCCR=1.0 とした場合，REDICと最適解の近似率は 90.97%，後続タスクの処理時間を考慮した場合，Path2~6 全て同値となる 91.28%となった。これは，通信遅延時間が各タスクの処理時間と比べると全体的に低く設定されているため，各タスクはPEに並列に割当てられ，通信遅延をある程度許可したスケジュールが最適解となっているためだと考えられる。CCR=2.0, 3.0 でも各下限値に顕著な違いは見られなかった。CCR=2.0 とした場合，REDICによる最適解との近似率は 81.00%，後続タスクの処理時間を考慮した場合は，最大でPath3~6 が 82.64%を示した。CCR=3.0 では，REDICによる最適解との近似率は 76.31%，後続タスクの処理時間を考慮した場合は，最大でPath4~6 が 79.65%を示した。

CCR=5.0, 10.0 とした場合は各下限に変化が見られた。CCR=5.0 では，REDICによる最適解との近似率は 70.31%，後続タスクの処理時間を考慮した場合，Path2 では，74.40%，Path3 では，76.81%，Path4 では，77.41%，Path5 では，77.67%，Path6 では，77.82%となった。CCR=10.0 では，REDICによる最適解との近似率は 65.61%，後続タスクの処理時間を考慮した場合，Path2 では，71.62%，Path3 では，77.19%，Path4 では，80.33%，Path5 では，82.55%，Path6 では，83.08%となった。最も顕著な違いが見られたCCR=10.0 では，通信遅延時間が各タスクの処理時間よりも全体的に非常に高くなるため，最適解が図 5 の逐次実行に近い割当てスケジュールとなっているためと考えられる。

CCR値の上昇につれて下限値と最適解との差は大きく開いてしまう。特に，REDICと最適解とではCCR=10.0 で比較した場合，34%もの差が出てしまう。この問題に対して，後続タスクの処理時間を考慮すれば，最適解との差を 17%まで縮めることに成功した。

CCR=1.0 の場合，通信遅延時間が全体的に低く設定されている為，後続タスクを当該タスクと同PEで処理せず別PEに並列に処理するスケジュールが最適解と成り易いため両下限値に大きな違いは見られなかった。しかし，60 個のタスクグラフの内，6 個のタスクグラフが，Path2 とした場合でも後続タスクの処理時間を考慮した方が下限値の向上を図れた。改善された下限値はREDICと比較した場合，最大で 5.36%，平均で 3.11%向上している。従って，通信遅延時間が低い問題でも効果があると考えられる。

4.5　探索中の下限値更新

通信遅延を考慮したCP長を生成する場合，PE数を限定しない環境で，全ての通信の発生を考慮しなければならない。これは，それ自身が組合せ最適化問題となってしまい，有効時間内で求めることは不可能である。しかし，現在の通信遅延を考慮した下限値をより正確なCP長に近づける場合，通信の発生を考慮した探索が必要となる。従って，探索中にRTからその直接後続タスクを処理するのに最低限必要な通信遅延を発見し，それをRTの下限値に組み込む。この場合，REDIC法と同じ，直接後続タスクの通信遅延のみとなってしまうため，あまり改善が見込めるとは思えない。しかし，REDIC法で発見できる通信遅延は図 11 のようなfork型のtree構造の場合であり，図 12 のようなjoin型のtree構造で必要な通信遅延は発見することができない。更に，探索中に下限値を更新することはその時点のスケジュールからエンドノードまでのより正確な最低限必要とされる経路長を発見することができ，これをタスクのプライオリティとして用いれば，経路長の長いタスクから優先的にPEへ割当てることを可能にする。これは暫定解が更新される確率の高い分枝操作が行える。従って，RTの下限値を更新することを試みる。

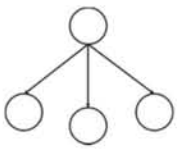

図 11　fork tree

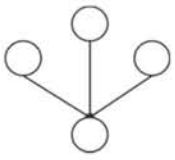

図 12　join tree

4.5.1　探索中の下限値更新アルゴリズム

下限値更新はDF/IHSのRT内のタスクに対して行われる。RTの更新後，RTの直接後続タスクを実行する際に最低限必要な通信遅延を取り込むことで下限値更新を行う。下記にアルゴリズムを示す。

① RT 内の各タスクの最早開始時間を決定する。この値は全ての PE に割り当てる組合せを考慮することで算出する。

② RT 内の各タスクの直接後続タスクの最早開始時間を求める。この値は各直接後続タスクを REDIC 法に掛けることで算出する。

③ 発生した通信遅延を組込むことで下限値を更新する。

このアルゴリズムで図 13 を用いて下限値更新を行う。

まず全てのRTの最早開始時間と最早完了時間を算出する。こちらは，REDIC法による算出ではなく，現在までのスケジュールを反映させるため，全てのPEに割当てた場合に，実際に最早開始時間・最早完了時間はどのタイミングかを決定する。図 13 のグラフを例にするとタスク 3，タスク 4 の最早開始時間・最早完了時間は図 14 のようになる。

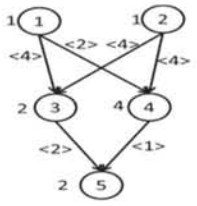

図 13　部分タスクグラフ

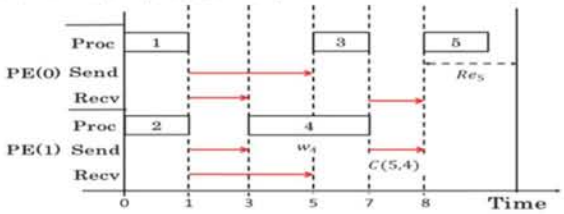

図 14　部分スケジューリング

図 14 より，タスク 3 の最早開始時間は時刻 5，タスク 4 は時刻 3 となる。ここからRTの直接後続タスクの最早開始時間・最早完了時間を求める。これにはREDIC法による計算から求める。これは，当該タスク(図 13 ではタスク 5)にとって最も早く処理を開始するために必要な通信遅延を算出する。図 14 のタスク 4 からタスク 5 に発生する通信遅延がタスク 5 を最も早く処理を開始するために必要な通信遅延となる。ここから下記の評価を行い，タスク 4 の現在の下限値より右辺の方が大きければ，タスク 4 の下限値を更新する。

$$Re_4 < w_4 + C(5,4) + Re_5$$

仮にタスク 5 を実際の割当てでPE(1)に割当て，タスク 4 からタスク 5 の通信遅延が発生しなくとも，その割り当て方はタスク 5 にとって最も早く処理を開始できるタイミングではない。従って，タスク 5 を経由しエンドノードに至るパス長は伸びてしまうため，最低限必要なパスとはならない。

ここで，タスク 3，タスク 4 の割当て方が各々最早開始時間・最早完了時間でない場合を考える。まず図 15 を例にとる。

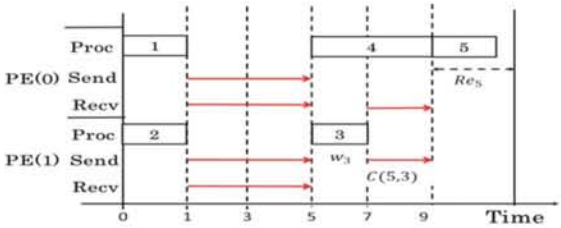

図 15　部分スケジューリング

図 15 よりRTであるタスク 3，タスク 4 が最早開始時

間・最早完了時間に割当てを受けていなければ，少なくともタスク5は図14の時刻よりも早く処理を開始できることはない。

この下限値更新は先行タスクのスケジュールに依存するため，先行タスクの割当て方が変更される毎に必要とされる通信遅延が変更される。従ってその都度下限値を更新する必要がある。

4.5.2　探索中の下限値更新の評価

探索時の下限値更新を行うスケジューラと下限値更新を行わないスケジューラとで性能差を示す。評価関数は以下とする。

- 最適解発見までの探索時間削減率

評価を行うタスクグラフのパラメータは，タスク数:20，タスク処理時間：1～20 の一様乱数，通信遅延時間:CCR=1.0，直接先行・後続タスク数：1～3 の一様乱数とした。このパラメータを所持したタスクグラフを40題用意した。このタスクグラフを4PEに割当てるスケジューリング問題を解いた。実験環境は4.3.2項と同様である。

探索時の下限値更新はRTが更新される毎に行われる。従って，下限値更新を行って枝切りが行われない限り，下限値更新の計算自体が無駄な計算時間となり，探索時間を延ばしてしまう場合がある。この問題に対して，評価用スケジューラの限定操作の評価手順を利用することで無駄な計算を減らす工夫を行う。評価用スケジューラはRTが更新される毎に枝切りを行うか評価するが，まず未割当てタスクの下限値を使用した枝切り，未割当てタスクの処理時間の総数をPE数で除算した値による枝切りの評価を行う。この評価は非常に少ない計算量で行えるため，この時点で枝切り可能な場合は下限値更新を行わない。もしこの2つの評価で枝切りが行えない場合，次にREDIC法を使用することで未割当てタスクの最早開始時間を算出し，その値から再度枝切りの評価を行う。下限値更新はこの未割当てタスクの最早開始時間を求める計算と同時に行う。この場合，RTのみ全てのPEに割当てた場合の組合せを考慮することで最早開始時間を求めRT以外の未割当てタスクは全てREDIC法の計算から最早開始時間を求める。従って，4.5.1.項で示した下限値更新アルゴリズムの①のみと下限値を更新するか否かの評価式を評価用スケジューラに加えるだけで良く，非常に少ない計算量で下限値更新を行うことが出来る。

4.5.3　評価結果

探索時の下限値更新を行うスケジューラと下限値更新をしないスケジューラとで評価を行った。どちらのスケジューラも元の下限値の値はREDIC法によって求められた値となっている。この評価では，下限値更新を行うことで最適解発見までの探索時間がどの程度削減されるのかを評価した。縦軸を探索時間削減率，横軸をタスクグラフ番号としている。結果を図16に示す。

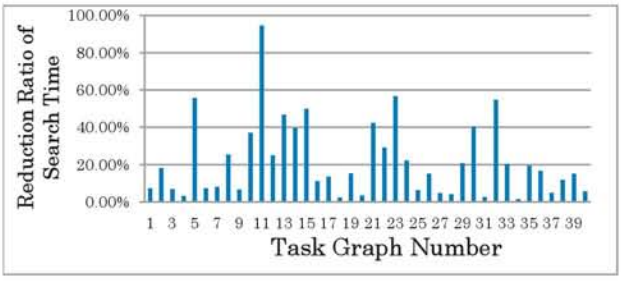

図16　探索時の下限値更新における探索時間削減率の評価

図16の結果より，平均で21.85%，最大で94.68%の探索時間を削減することに成功した。問題ごとに削減率に差があるのはタスクグラフの構造によりREDICの精度に差が出てしまうことや下限値更新があまり行われなかったためだと考えられる。この下限値更新は通信遅延のみしか下限値に組み込めないため，通信を発生させず各タスクを特定のPEで逐次に実行するようなスケジュールをとる場合，下限値更新はあまり行われない。更に，図12 のjoin-tree型が少ないタスクグラフでも同様にREDICよりも優れた下限値を発見することは難しい。しかし，今回評価した全てのタスクグラフで探索時間を削減することができた。これは，少なからずどのタスクグラフでも下限値を更新することで，新たに枝切りを行えたことと，下限値更新自体の計算を非常に少ない時間で行うことが出来たためだと考えられる。従って，下限値更新を行うことでスケジューリング効率を向上させることができたと考えられる。

5. 通信遅延を考慮した最適化スケジューラの構築

前章までの下限値生成を最適化スケジューラに組込む。下限値生成は，探索を行う前処理の工程と探索時の工程の2つに分けて行う。まず前処理の工程では，REDIC法に加えてPE数を考慮した下限値生成，後続タスクの処理時間を考慮した下限値生成を行う。探索時の工程では，探索時の下限値更新と並列度paraを用いたPE数を考慮した下限値生成を行う。これは，昨年度まではREDIC法による探索前のスタティックな下限値生成のみであったのに対し，新たに改良された下限値生成を用いれば，常に探索時の下限値更新というダイナミックな下限値生成要素も加えられる。更に，下限値更新で使用される元の下

限値はREDIC法に加え，PE数と後続タスクの処理時間も考慮された値となるため，REDIC以上の値が使用される。

6．実験

本章では，前述した3つのアルゴリズムを追加したスケジューラと昨年度までのスケジューラとで評価を行う。評価関数を次に示す。

① 最適解発見までの探索時間削減率の評価

② 指定実行時間内の探索での初期解からの暫定解向上率

①では，最適解を発見するまでに費やした探索時間がどの程度削減することができるかを評価する。この評価を行うタスクグラフのパラメータは，タスク数：20，タスク処理時間：1～30，直接先行・後続タスク数：1～3とした。このタスクグラフから2，3，4PEに割当てるスケジューリング問題を20個解き評価を行う。

②は，大規模問題を指定実行時間内で探索を行うことで初期解から暫定解がどの程度良くなるのかを評価する。この評価を行うタスクグラフパラメータは，早稲田大学笠原研究室の標準タスクセット[8]のタスク数 50 とし，CCR=1.0 と設定した。このパラメータを所持したタスクグラフを30題用意し，3PEに割当てるスケジューリング問題を解く。指定実行時間内での探索では，下限値を使用した枝切りの精度を上げる以外に分枝操作で使用する各タスクの割当て優先度も重要となる。これは探索枝の順序に変化を与え，探索PEが指定実行時間内に解ける子問題にどれだけ優良解を集められるかに繋がる。従って，この評価では新しい下限値生成を追加したスケジューラにこの下限値を各タスクの割当て優先度に用いる。

使用したマシンは4.3.2.項と同様となっている。この環境より12コアによる並列探索を行う。

7．評価

7.1　最適解発見までの探索時間削減率の評価

昨年度までのスケジューラと今回新しい下限値生成を追加したスケジューラとで探索時間削減率の評価を行った。縦軸に探索時間削減率，横軸を割当てPE数とした。20個のタスクグラフにPE数2，3，4としたスケジューリング問題を解き，各々の結果を示す。

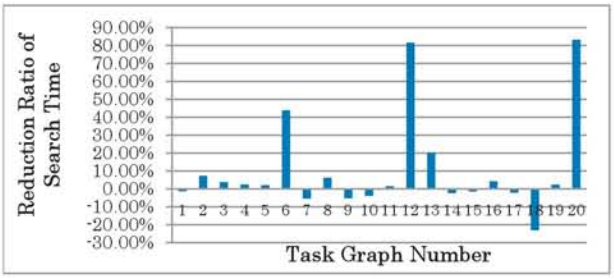

図17　割当てPE数を2とした問題の最適解発見までの探索時間

図17は割当てPE数2とした場合の評価結果である。平均で 10.70%，最大で 83.27%の探索時間を削減することができた。探索時間が延びてしまった問題は，探索中にPE数を考慮した下限値生成を行ったが，それが枝切りに繋がらなかったためだと考えられる。PE数を考慮した下限値生成を探索時に行う場合，RTが更新される毎に$O(V * Re_s^2)$の計算量が費やされてしまう。この計算時間を払った上で枝切りへと繋がれば探索時間を削減できる可能性が高まるが，もしできなければ無駄な計算時間となり，探索時間を単純に延ばしてしまう。しかし，ここで最も高い探索時間削減率を算出したタスクグラフ 20 番を例に取り，探索時にPE数を考慮した下限値生成を行うか行わないかで実験をしてみる。この実験では単一PEで探索を行う。結果を表2に示す。

表2では，Search Timeは実際の探索時間，Branch Numは探索した探索枝の数となっている。この結果から探索時にPE数を考慮することで劇的なスケジューリング効率の向上を確認できる。

表2　探索時のPE数を考慮した下限値生成による性能評価

アルゴリズム	探索時間(秒)	分枝数
探索時にPE数を考慮した下限値生成を行わない	5.1115	5252051
探索時にPE数を考慮した下限値生成を行う	0.874	122749

次に割当てPE数3とした場合の探索時間削減率の評価結果を示す。

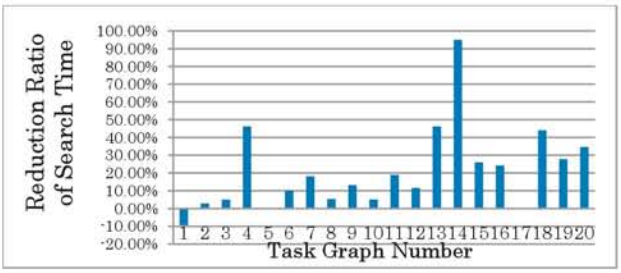

図 18　割当てPE数を 3 とした問題の最適解発見までの探索時間削減率の評価結果

図 18 は, 割当てPE数 3 とした場合の評価結果である。平均で 21.31%，最大で 94.94%の探索時間を削減することができた。探索時間が延びてしまった問題は，割当てPE数 2 とした場合の考察と同様だと考えられる。図 18 の中で棒グラフが出ていないものはミリ秒単位の誤差となったため，このレベルだと平均値に冗長な変化を与えてしまうので 0%とした。

次に割当てPE数 4 とした場合の探索時間削減率の評価結果を示す。

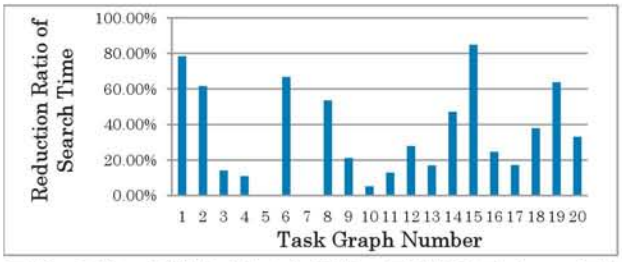

図 19　割当てPE数を 4 とした問題の最適解発見までの探索時間削減率の評価結果

図 19 は, 割当てPE数 4 とした場合の評価結果である。平均で 30.00%，最大で 84.86%の探索時間を削減することができた。図 19 の中で棒グラフが出ていないものは割当てPE数 3 とした場合の考察と同様である。

割当てPE数 3，4 とした場合は，実験で使用した 20 題のタスクグラフにとって十分な並列処理性能を期待できる。この場合，PE数を考慮した下限値生成による下限値向上は望めないが，後続タスクの処理時間を考慮した下限値生成と探索時の下限値更新による枝切り効果が期待できる。割当てPE数 4 とした場合は，PE数を十分以上に所持してしまっている可能性が高く，冗長な組合せを生成してしまうことがある。このような場合は探索時の下限値更新による枝切り効果が高く結果に反映される。通信遅延を考慮した最適化スケジューラはより並列性能が得られる組合せから優先して探索を行う。これは割当てPE数を十分以上に所持している場合，冗長な通信を発生させる組合せを優先して探索することに繋がる。従って，スケジューリング効率を落としてしまうことがある。しかし，下限値更新によりこの通信遅延を下限値に組込むことで枝切り効果を上げることができる。図 19 の結果からも下限値更新により枝切りの精度が向上したと考えられる。

7. 2　初期解からの短縮率

昨年度までのスケジューラと今回新しい下限値生成を追加したスケジューラとで大規模問題を指定実行時間内で解いた場合の初期解からどの程度暫定解が向上されるのかを評価した。探索時間は 600 秒間とした。縦軸に暫定解向上率，横軸をタスクグラフ番号とした。

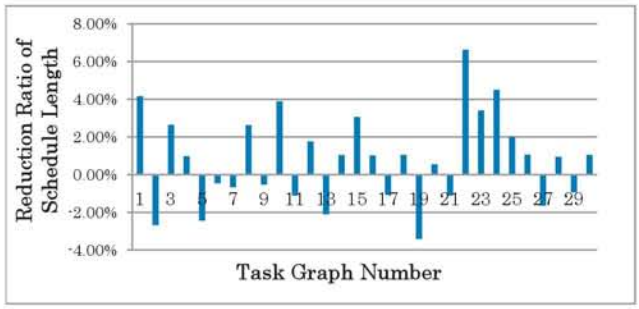

図 20　指定実行時間内の探索での初期解からの暫定解向上率

図 20 より，最大で 6.63%，平均で 0.81%となった。タスクの優先度の変化により短縮率が下がってしまった問題もあるが，全体平均ではスケジューリング効率の向上を確認できる。

大規模問題から最適解を得る場合，有効な下限値を使用した枝切り以外に，分枝操作によって探索の早期にどれだけ最適解により近い暫定解を算出できるかが重要となる。分枝限定法は分枝操作によって生成された子問題を全て探索しない限り，別の子問題の探索を開始することが出来ない。大規模問題の場合，探索空間の初期で下限値により近い上限値を得られないと, 枝切りが進まず, 優良解を発見できそうにない子問題から抜け出せなくなる。DF/IHS法ではこの問題に対して，CP/MISFのヒューリスティックを各タスクのプライオリティレベルとし，この優先度に従った探索を行うことで探索初期の空間に優良解を集める手法を取っている。しかし，これに通信遅延を考慮した場合，CP/MISFのアルゴリズム自体が通信遅延を考慮していないため，この優先度に従って探索をすると通信遅延を考慮する前と同様の期待を得ることはできない。今回の実験ではこの優先度にREDICとPE数を考慮した下限値生成と後続タスクの処理時間を考慮した下限値生成によって算出された下限値を使用した。更に，探索時の下限値更新を行いレディタスクの下限値が

変更された場合は，下限値の降順に優先度を変更している。結果としては，実験に使用した30題のタスクグラフの内15題が暫定解を更新することに成功している。勿論，もう半分の 15 題は昨年度までのスケジューラの方が指定実行時間内により良い解の探索に成功しているが，こちらは今回下限値生成を追加したスケジューラが下限値生成の時間に手間取り探索が上手く進まなかったのではなく，新しく設定した各タスクのプライオリティレベルでは探索空間の初期に優良解を発見でそうな子問題を生成できなかったためだと考えられる。今回設定したプライオリティレベルでは各タスクにエンドノードまでの経路上に存在する全ての通信を考慮したわけではないので，本来下限値に加えるべき通信遅延時間を無視した値となり，間違った優先度情報を与えてしまったためである。従って，今後はこの各タスクのプライオリティレベルを向上させることでスケジューリング効率を上げられると考えている。

8．考察

今回追加した下限値生成アルゴリズムにより，どちらの評価関数でもスケジューリング効率を向上させることが出来たと考えられる。これは，探索前のスタティックに求められる下限値が常にREDIC以上の値を算出でき，それを探索時の下限値更新に使用することで昨年度までのスケジューラよりも枝切りの精度を落とすことがないためだと考えられる。従って，今回追加した下限値生成自体の計算時間が探索時に影響を与えない限りスケジューリング効率を向上させることが出来る。

9．おわりに

本稿では分枝限定法で使用する下限値に対して，PE数を考慮した下限値生成，後続タスクの処理時間を考慮した下限値生成，探索時の下限値更新という3つの下限値生成アルゴリズムを提案した。これを昨年度までのREDICと併用することで，分枝限定法の探索効率を向上させることに成功した。

謝辞

本研究は科研費（基盤研究(C)21500041）の助成を受けたものであることをここに記し，謝意を表します。

参考文献

[1] M.R.Garey, D.S.Johnson, “Computers and Intractability : A Guide to the Theory of NP-Completeness”, W.H.Freeman;First Edition,1979

[2] Koichi Kurita, Masahiko Utsunomiya, Ryuji Shioda and Munenori Kai:“Development and Evaluation of Parallel Search Method to Solve Task Scheduling Problems on Account of Communication Overhead”, 情報科学技術フォーラム 2011

[3] Masahiko Utsunomiya, Ryuji Shioda and Munenori Kai:“Heuristic search based on branch and bound method for task scheduling considering communication overhead”, IEEE Pacific Rim Conference on Communications, Computers and Signal Processing, 2011

[4] T.C.Hu:“Parallel sequencing and assembly line problems”, Operations Research, Vol.9, No.6, pp.841-848, 1961

[5] C.V.Ramamoorthy, K.M Chandy and M.J.Gonzalez Jr.:“Optimal Scheduling Strategies in a Multi- processor System”, IEEE Trans.Computers, vol.21, No.2, pp.137-146, Feb.1972

[6] E.B.Fernandez and B.Bussell: “Bounds on the Number of Processor and Time for Multiprocessor Optimal Schedules”, IEEE Trans. Computers, Vol.22, No.8, pp.745-751, Aug.1973

[7] H.Kasahara and S.Narita:“Practial Multiprocessor Scheduling Algorithm for Efficient Parallel Processing”, IEEE Trans. Computers, Vol.33, No.11, pp.1023-1029, Nov. 1985

[8] Kasahara Lab. in Waseda Univ.: Standard Task Graph Set, http://www.kasahara.elec.waseda.ac.jp/schedule/

[9] T.Tobita and H.Kasahara:“A standard task graph set for fair evaluation of multiprocessor scheduling algorithms”, Journal of Scheduling, Vol.5, No. 5, pp.379-394, Oct. 2002