A-029

通信を考慮したタスクスケジューリング問題の探索解法のための高速化手法
Evaluation of Fast Search Methods to Solve Task Scheduling Problems Taking Account of Communication Overhead

宇都宮 雅彦† 塩田 隆二† 甲斐 宗徳†
Masahiko Utsunomiya Ryuji Shioda Munenori Kai

1. はじめに

タスクスケジューリングにおいて最適解を求めるためには、分枝限定法を用いて全探索を行うことが必要である。しかし、一般化されたタスクスケジューリング問題は、強 NP 困難な複雑度を持つ組み合わせ最適化問題であることが知られている。強 NP 困難な複雑度を持つ組み合わせ最適化問題では、問題サイズが大きくなるにつれて探索範囲が指数関数的に増加してしまうため、サイズの大きい問題ではその最適解を得ることが非常に困難となる。

このタスクスケジューリング問題を、分散メモリ型マルチプロセッサモデルにおける並列処理に適用すると、タスクの依存関係に応じてプロセッサ間に通信時間が発生する。したがって、プロセッサ間の通信時間を考慮したタスクスケジューリングでは、通信の組み合わせが強 NP 困難な問題をさらに複雑化させてしまう。

粗結合を含むマルチプロセッサ環境に対してより現実的なスケジュールを求めたいタスクスケジューラにとって、通信を考慮したタスクスケジューリングの探索時間の削減、および解の精度向上は必要不可欠である。そして、その実現のためには何らかのヒューリスティックなアルゴリズムによる工夫が必要である。

本研究の目的は、通信を考慮したタスクスケジューリングの探索手法のための高速化である。本発表では、内部の並列性を考慮して行うタスクのグループ化と、通信を考慮した下限値の導出法の２つのヒューリスティックアルゴリズムを提案し、その評価結果について報告する。

2. タスクスケジューリング問題と従来の解法

タスクスケジューリング問題とは、タスク単位の並列処理時間を最小にすることを目的とした、処理装置への割り当ての決定である。タスクとは、対象となる処理全体を分割した処理単位である。タスクをノード、先行制約を有向エッジで表した非循環有向グラフをタスクグラフと呼ぶ。タスクグラフの各タスクは、許された個数の処理装置に割り当てられ、並列に処理される。「一般形状の先行制約を持ち、各タスクの処理時間が異なり、プロセッサ数も任意」という、一般化されたタスクスケジューリング問題は、強 NP 困難な計算複雑度を持つ。

このようなタスクスケジューリング問題に対して、笠原らが提案した CP/MISF(Critical Path / Most Immediate Successors First)法と DF/IHS(Depth First/Implicit Heuristic Search)法がある。[1] DF/IHS 法は、組み合わせ最適化問題に対して最も有効な探索手法である分枝限定法に、CP/MISF 法におけるヒューリスティック効果を取り入れた、全探索アルゴリズムである。

†成蹊大学理工学研究科理工学専攻 Graduate School of Science and Technology, Seikei University

3. 通信を考慮した下限値

タスクスケジューリングでの枝切り操作とプライオリティレベルの設定には、想定できる任意のタスクから出口タスクまでの最短距離（下限値）を使用する。従来使用する下限値は各タスクの処理時間だけを元に算出しており、タスク間の通信による遅延は考慮されていなかった。通信を考慮した、より精度の良い下限値を使用することができれば、これを枝切りやプライオリティレベルに使用することでスケジューラの性能向上が期待できる。本研究では、通信を考慮したタスクの下限値の算出方法を考案した。

3.1 下限値計算の条件

下限値計算では、下限値を求めるタスクが終了してから出口タスクまでの最短経路を求める必要がある。しかし、後続タスクのすべての通信状況を考慮することは、それ自体が組み合わせ最適化問題になるため困難である。そのため、下限値の計算が複雑化しないような以下の条件を設定した。

①直接後続タスクの下限値を元に計算する
②すべての処理装置はアイドル状態とする
③処理装置の数を制限しない
④後続タスク同士の先行関係を考慮しない

この条件を用いて計算を簡略化することで、従来に比べて精度の高い下限値が単純な計算で算出可能となる。

3.2 下限値計算

下限値の値は、下限値を求めたいタスクの処理後から出口タスク処理までの最低限必要な処理時間と、下限値を求めたいタスクの処理時間を合算した値となる。

下限値計算で使用する値を以下に記す。

A：下限値を求めるタスク
NCPE(Non Communication Processor Element)：タスク A を処理したプロセッサ
CPE(Communication Processor Element) ：タスク A を処理していないプロセッサ
t ：任意の直接後続タスク
time(t) ：タスク t の処理時間
com(t) ：タスク A からタスク t への通信時間
lb(t) ：タスク t の下限値
lb'(t) : lb(t) - time(t)
pass(t) ：タスク A 処理後タスク t を通る出口タスクまでに必要な処理時間

絶対パス長：タスク A の直接後続タスクの、プロセッサに対する割り当てをある組み合わせにしたときの、タスク A の直接後続タスクの pass の最大値

3.2.1 NCPE で全てのタスクを処理する場合

NCPE にすべての直接後続タスクを割り当てる場合を考える。タスク A 終了後の最低限必要な処理時間は、lb’の降順にタスクを処理した時の絶対パス長となる。

NCPE で全タスクが処理される場合、タスク A 処理後に lb’の降順で t 番目に処理するタスクをタスク t とする。

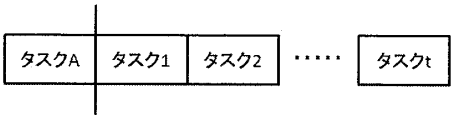

図 3.1 NCPE におけるタスクの処理位置

また、タスク A の直接後続タスクを NCPE に割り当てた時、自身の pass(x)が絶対パス長となるタスクを x 、x 以前に処理するタスクを w 、x 以後に処理するタスクを y と呼ぶ。

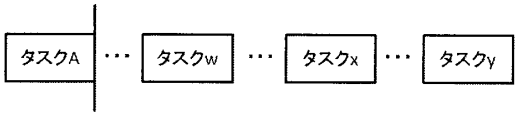

図 3.2 NCPE におけるタスク w、タスク x、タスク y

タスク A 処理後の絶対パス長は

pass(x)= time(1)+…+time(x)+lb’(x)

タスク x を後方で処理すると time(1)～time(x)が増加することになるので

①タスク x を後方で処理すると絶対パス長は増加

また、以下の 2 つの式より

② w と y の処理順を入れ換えると絶対パス長は増加

pass(w)=time(1)+…+time(x)+…+time(y)+lb’(w)

lb’(w) > lb’(x)

また、タスク x の処理を前方で処理すると元のタスク x の順で処理するタスク w が生じるので

pass(w)=time(1)+…+time(x)+lb’(w)

lb’ (w) > lb’ (x)

③タスク x を前方で処理すると絶対パス長は増加する

①、②、③より NCPE において lb’の降順にタスクを処理すると絶対パス長が最小となる。

3.2.2 全てのプロセッサを使用する場合

次に CPE も使用して処理する場合のタスク A 終了後の最低限必要な処理時間を求める。CPE でタスク t を処理するとそのパスは pass(t)=com(t)+lb(t) となる。

そのため、以下の手順でタスク A 終了後の最低限必要な処理時間を求める。

①NCPE で処理しているタスクの中で com(t)+lb(t)が最小となるタスク t を探索。

I. NCPE で処理するタスクが 1 つなら、そのタスクのパスが求める値となる。

図 3.1 タスク t の選択

②NCPE で処理するタスクの pass の最大値と com(t)+lb(t)を比較。

I. com(t)+lb(t)が大きいなら NCPE で処理するタスクの pass の最大値が求める値。

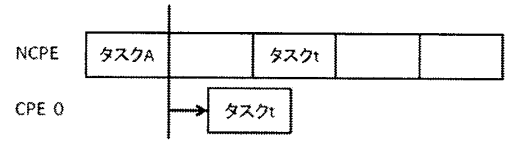

図 3.2 タスク t を処理するプロセッサの選択

③タスク t を CPE で処理することとし、NCPE で処理するタスクからタスク t を除いて NCPE で処理するタスクの pass の最大値を再計算する。

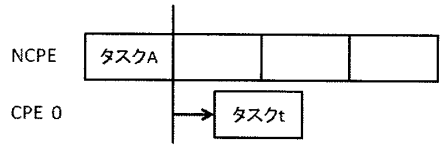

図 3.3 タスク t を NCPE から削除

④再計算した NCPE で処理するタスクの pass の最大値と com(t)+lb(t)を比較。

I. com(t)+lb(t)が大きいなら com(t)+lb(t)が求める値。

II. com(t)+lb(t)が小さいなら①に戻る。

以上①～④の手順で求めた値にタスク A の処理時間を加えた値がタスク A の下限値となる。

3.3 下限値の使用方法

求めた下限値をプライオリティレベルとして使用することはヒューリスティックであるため、従来の下限値より探索効率が必ず良くなるわけではない。そこで、従来の通信時間を考慮しない下限値と考慮した下限値をプライオリティレベルとして使用した探索でそれぞれヒューリスティック的に最も良いとされる解を算出し、良い解を算出した下限値をプライオリティレベルとして使用することにした。

また、最短経路の近似値として精度が上がっているため枝切りの下限値として使用した。

3.4 実行

3.4.1 評価方法と実行環境

スケジューリング効率の評価は 2 種類行った。

1 つ目の評価は 2 つの下限値を用いて評価用タスクグラフを全探索した際の探索時間を比較した（以下、探索時間評価と呼ぶ）。従来の下限値より通信を考慮した下限値を使用した方が、全探索時間がどのくらい削減されるかでスケジューリング効率の向上を評価する。

2 つ目の評価は 2 つの下限値を用いて評価用タスクグラフのスケジューリングを行い、設定した時間で求まる暫定解を比較した（以下、暫定解評価と呼ぶ）。従来の下限値より通信を考慮した下限値を使用した方が、設定した時間で求まる暫定解がどのくらい短縮するかでスケジューリング効率の向上を評価する。

2 つの評価は以下のようになっている。

探索時間評価タスクグラフ
- 本研究室のランダムタスクグラフジェネレータで生成したタスクグラフ
- タスク数 21
- タスク処理時間 1～30
- 通信時間 1～60
- 後続タスク数 1～4　先行タスク数 1～4
- タスクグラフ 80 個を生成。

探索時間評価方法
- 従来の下限値を使用してタスクグラフの 4 プロセッサに対するスケジューリングを行う。
- 今回提案する下限値を使用して探索時間評価用タスクグラフの 4 プロセッサに対するスケジューリングを行う。

以上のスケジューリングでの探索時間を比較して、今回提案する通信を考慮した下限値を使用した場合のスケジューリング効率の向上を評価する。

暫定解評価用タスクグラフ
- 早稲田大学笠原研究室の標準タスクセット[2]
- タスク数 100
- CCR と呼ばれる比を用いて通信時間を付加した。[3] CCR を 10 と設定しタスクグラフを 30 個生成。

暫定解評価方法
- 従来の下限値を使用してタスクグラフの 4 プロセッサに対するスケジューリングを 60 秒の探索時間で行う。
- 今回提案する下限値を使用して暫定解評価用タスクグラフの 4 プロセッサに対するスケジューリングを 60 秒の探索時間で行う。

以上のスケジューリングで求まった解を比較して、今回提案する通信を考慮した下限値を使用した場合のスケジューリング効率の向上を評価する。

使用したマシンの仕様は以下である。

CPU : Intel(R) Xeon(R) X5550 @ 2.67GHz x 2

OS　: Linux

RAM　: 12GB

これら 2 つの評価は並列探索における相乗効果も期待できるため、上記のマシンで 8 コアを使用して並列実行した。

3.4.2 探索時間評価の実行結果

図 3.6 がタスクグラフ番号、縦軸が探索時間削減率となっている。

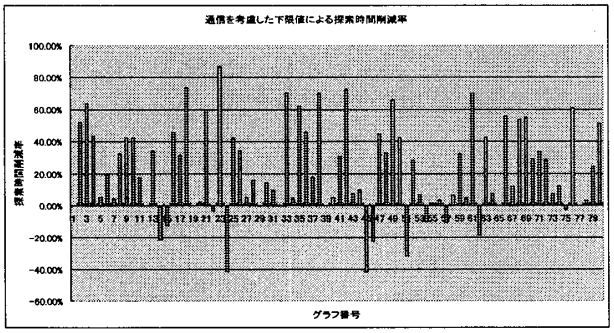

図 3.4 探索時間評価の結果

全探索時間の削減率はタスクグラフに依存するが、通信を考慮した下限値を使用して全体平均で全探索時間が 22.15%削減された。

3.4.3 暫定解評価の実行結果

図 3.7 がタスクグラフ番号、縦軸が暫定解の短縮率となっている。

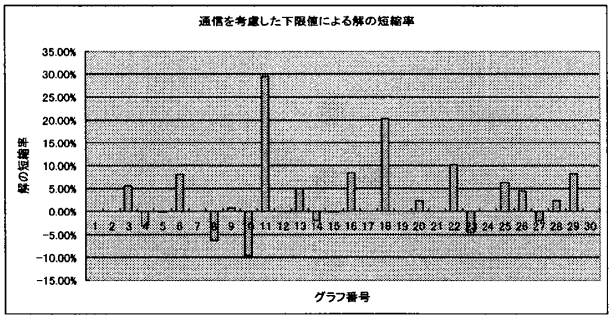

図 3.5 暫定解評価の結果

暫定解の短縮率はタスクグラフに依存している。短縮率の幅は-10%～20%。通信を考慮した下限値を使用して全体平均で 2.78%求まる解が短縮した。

3.5 評価と考察

3.5.1 探索時間評価

全体を通して探索時間が削減されている。一部では探索時間が長くなったが、一部では 90%以上探索時間が削減されるタスクグラフも存在した。全体的にも 17~20%近い探索時間の削減がされているため、スケジューリング効率の向上になっていると判断できる。

3.5.2 暫定解評価

解の短縮が見られるタスクグラフは限られていて、一部で求まる解が悪くなったが劇的に悪くなる例は稀であった。設定した時間内で探索できる解が向上傾向にあり、タスクの処理時間に対する通信時間の割合が大きいほど暫定解の変化幅が大きくなっていた。以上のことより通信を考慮した下限値の使用で解の短縮傾向が見られ、スケジューリング効率の向上になっていると判断できる。

3.5.3 考察

スケジューリング効率向上の幅がタスクグラフに依存しているが、評価用タスクグラフを全体的に見ても決められた時間内での暫定解が短縮傾向にあり、全探索時間は減少傾向にある。このことからも通信を考慮した下限値を使用することでスケジューリング効率が向上しているといえる。

4. 自動マクロタスク化アルゴリズム

通信を考慮したタスクスケジューリングの探索時間の短縮、及び精度の向上を行うための新たなヒューリスティックアルゴリズムとして、自動マクロタスク化アルゴリズムを開発した。

4.1 マクロタスク

タスクグラフ中のマクロ化可能なタスク群をまとめ、元のタスクグラフにおいて 1 つのタスク X として取り扱

う時、このタスク X をタスク群に対するマクロタスクと呼ぶ。

あるタスク群について、タスク群の開始点となるタスクをタスク A、終点となるタスクをタスク B とする。このとき、タスク群の始点タスク A 以外のタスクについて、タスク A を経由せずにそれらのタスクへ到達可能なエッジを流入エッジと呼ぶ。また、タスク群の B 以外のタスクについて、タスク B に到達できない経路を流出エッジと呼ぶ。

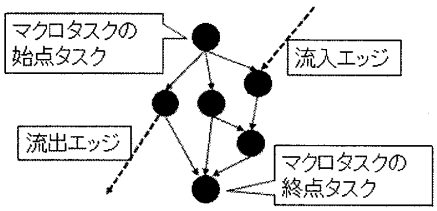

図 4.6 マクロタスク

今回開発した自動マクロタスク化アルゴリズムがマクロ化可能であると判断するタスク群は、タスクグラフ上の任意のタスク A, B について、タスク A から分岐、任意のタスク n 個を任意の経路で経由し、タスク B で収束、かつ流入エッジ・流出エッジが存在しないタスク群である。また、マクロタスクはマクロタスクを内包することができ、一つのタスク集合には複数のマクロタスクを内包することができる。

4.2 マクロタスクを含むタスクグラフを処理するためのスケジューラの改良

マクロタスクを含むタスク集合を処理するために従来のスケジューラへ改良を行い、占有プロセッサ数が 2 以上のタスクを含むタスクグラフを、取り扱うことができるようになった。占有プロセッサ数が 2 以上のタスクについて、そのタスクを処理するためには 2 以上の必要な数のプロセッサが ready 状態になる必要がある。

部分スケジューリングの結果、そのマクロタスク内の最適解を得るために必要となるプロセッサ数が値 a である場合、この占有プロセッサ数を a として、マクロタスクを内包するタスク集合へと反映する。a は 2 以上の値になる可能性がある。

4.3 マクロタスクの成立判定

あるタスク集合がマクロタスクとして成立するための条件は、そのタスク集合に流出エッジおよび流入エッジが存在しないことである。

4.3.1 流出エッジの可能性の消去

まず、対象のタスク群について、群中のとあるタスクをタスク x としたとき、タスク群中の始点タスク以外のすべてのタスクについて、次の 2 つの値を算出する。

I. 始点タスク A から生じる経路がタスク x に到達するまでに発生した分岐数
II. 始点タスク A から生じる経路がタスク x に到達した回数

調査の結果、I と II の値が等しくなったタスクを、マクロタスクの終点タスク B として仮決定する。

以上の手続きにより、求めた始点タスク A と終点タスク B によるタスク集合に流出エッジが存在しないことを保証する。

4.3.2 流入エッジの可能性の消去

前項で求めたタスク集合の、始点タスク A 以外のすべてのタスクについて、各直接先行タスクが始点タスク A から到達可能であるかを調査する。到達可能である場合、始点タスクを A、終点タスクを B とするタスク集合に流入エッジが存在しないことを保証する。

4.4 マクロ化対象タスク群抽出ルーチン

マクロタスクになる可能性のあるタスク群を、マクロタスク成立判定アルゴリズムに渡していくことで、タスクグラフのマクロ化を行うルーチンである。タスクグラフの始点タスクから始め、直接後続タスクに対して順次マクロ化を行うことで元のタスクグラフ全体のマクロタスク化が完了する。

マクロタスクとして判定できる可能性のあるタスク群は、直接後続タスク数が α 以上のタスクから、β だけ先までのタスク全てである。この範囲をアルゴリズムの探索範囲とし、以降ではこの β を探索上限距離と呼ぶ。あるタスクにとって自身から β だけ先までのタスク集合の中の可能なマクロタスク化がすべて終了した時点で、そのタスクを始点タスクとしたマクロタスクの成立判定を行う。以下にそのプロセスを示す。

① 始点候補となるタスクを 1 つ選択する
② 選択したタスク以降で探索上限距離までの範囲に、直接後続タスク数が α 以上となるタスクが存在するかを調べる
③ ②が真である場合、そのタスクを始点タスクとしてこの抽出アルゴリズムを呼び出し、②の判定が偽になるまでこれを繰り返す
④ 選択範囲をマクロタスク成立判定アルゴリズムに渡す
⑤ この再帰的な処理によって一連のマクロタスク化が 1 つ終了するごとに、元のタスクグラフの始点タスクから探索を再開する

4.5 自動マクロタスク化アルゴリズムの実行結果

4.5.1 実行環境

使用マシンの仕様は 3.4 で紹介したものと同じである。

タスクグラフの生成には当研究室のランダムタスクグラフジェネレータを使用した。

4.5.2 探索時間削減の評価

ランダムに生成したタスクグラフに対し、自動マクロタスク化アルゴリズムの適用前と適用後で、探索時間の比較を行った。表 4.1 にその結果を示す。

表 4.1 アルゴリズム適用による探索時間の削減

タスクグラフ	自動マクロタスク化	探索時間 (sec.)	マクロタスク化に要した時間(sec.)
例1	適用前	18.860	
	適用後	0.890	0.070
例2	適用前	268.640	
	適用後	21.540	0.010
例3	適用前	1425.800	
	適用後	300.010	0.040

タスク数 20、最大直接先行タスク数 2、最大直接先行タスク数 2、の設定で生成を行った。また、割り当てプロセッサ数には自動マクロタスク化を行う前のタスクグラフで最適解を算出できる数を指定した。

自動マクロタスク化アルゴリズムの適用によって、全てのタスクグラフで探索時間の削減が見られた。また、自動マクロタスク化アルゴリズムの適用にかかる時間は、どのケースにおいても非常に短時間であった。

4.5.3 最適解の求解に必要なプロセッサ数

タスクグラフに自動マクロタスク化を適用することで、クリティカルパス長を求めるために必要なプロセッサ数が変化するかを調査する。元のタスクグラフ（クリティカルパス長 170）と自動マクロタスク化適用後のタスクグラフについて、割り当てプロセッサ数を変化させてスケジューラが求める解と探索時間を比較した。表 4.2a、4.2b に結果を示す。

表 4.2a 元のタスクグラフの測定

割り当てプロセッサ数	求めた解	探索時間(sec.)
2	186	261.170
3	170	24.190
4	170	32.620

表 4.2 b 自動マクロタスク化適用後の測定

割り当てプロセッサ数	求めた解	探索時間(sec.)
2	186	261.170
3	170	24.190
4	170	32.620

使用可能プロセッサ数 3 での探索は、スケジュール結果に差が見られた。マクロタスク化を行っていないタスクグラフでは、解としてクリティカルパス長が算出され、求めた解が最適解であることがわかる。

使用可能プロセッサ数を 4 に増やして行ったスケジューリングでは、求めた解に差は無く、どちらもクリティカルパスを求めることに成功した。

4.5.4 マクロタスクのプロセッサ占有率による精度への影響

ランダムに生成したとあるタスクグラフ（クリティカルパス長 84）の中に 1 つだけ特にコストの高いタスク A を用意し、マクロタスクのプロセッサ占有率の違いが全体のスケジュール長に及ぼす影響を調べた。結果を表 4.3 に示す。

表 4.3 プロセッサ占有率が解へ与える影響

	求めた解	探索時間(sec.)
元のタスクグラフ	186	261.170
自動マクロタスク化アルゴリズム適用後	170	24.190
プロセッサ占有率の低いマクロタスク化を取り消し	170	32.620

マクロタスク化を行わなかった場合、解はクリティカルパス長である 84、探索時間は 11.860 秒であった。

自動マクロタスク化アルゴリズムの適用時には、形状として可能なマクロタスク化がすべて行なわれ、タスク A はマクロタスクに内包された。解は 125 となり、自動マクロタスク化を適用する前と比べ、約 395 倍速の探索時間で解を求めることができた。

タスク A がマクロタスクに内包されないようにマクロタスク化を一部に適用した場合、解は 86 となった。マクロタスク化を適用する前と比べ、約 148 倍速の探索時間で解を求めることができた。

4.6 評価と考察

自動マクロタスク化アルゴリズムの開発によって、マクロタスクとして判定可能なパターンを含むタスクグラフの探索時間を削減することが可能となった。削減の度合いにはばらつきがあり、削減度合いの大きいものでは従来に比べ 140 倍以上の探索時間の短縮につながっている。

自動マクロタスク化アルゴリズム適用後のスケジューリングは元のタスクグラフをそのままスケジューリングするよりも、最適解を求めるために多くのプロセッサを必要とする可能性がある。自動マクロタスク化後の探索時間は減少傾向にあることから、探索時間とプロセッサ数の間にトレードオフの関係が発生しているといえる。

パターンに一致しても、自動マクロタスク化を行わない方が良い場合が存在する。スケジュール結果に影響を与えているのはマクロタスクごとのプロセッサ占有率であるから、自動で行うマクロ化の判断材料として、プロセッサ占有率について閾値を設定することが考えられる。

5. おわりに

本発表では、通信を考慮した下限値の新たな算出方法と、自動でタスクグラフのマクロ化を行うアルゴリズムについて述べた。どちらも、ランダムに用意したタスクグラフに対してスケジューラの探索時間が改善される傾向がみられた。また、今後の改善指針についても触れ、より大規模なタスクグラフを高速に探索可能なタスクスケジューラの開発への課題とした。

本研究は実応用に向けたタスクスケジューラの開発である。実際のアプリケーションをタスクグラフ化した際には、単純にランダムな生成を行ったタスクグラフには見られないような特徴があるものと考えている。したがって、実応用のタスクグラフの特徴をとらえたアルゴリズムを開発することで、より実用的なスケジューラとして改善が可能と考えられる。

謝辞

本研究の一部は、文部科学省戦略的研究基盤形成支援事業の補助を受けて行ったことをここに記し、謝意を表します。

参考文献

[1]笠原博徳, ”並列処理技術”, コロナ社, 1991
[2]STG (Standard Task Graph Set), http://www.kasahara.elec.waseda.ac.jp/schedule/in¬dex.html 2010 年 1 月現在
[3]O.Sinnen, “Task Scheduling for Parallel Systems”, Wiley-Interscience (2007).