マルチメディア通信と分散処理 37－12
（１９８８．５．２０）

トーラス型マルチプロセッサシステム上での分枝限定アルゴリズム

川口 剛　　真栄田 保　　喜屋武 盛基

琉球大学 工学部

あらまし　トーラス型マルチプロセッサシステム上での並列分枝限定アルゴリズムを提案する。並列分枝限定アルゴリズムに関する研究は、負荷分散手法の容易さのため、主に共有メモリ型システムに対して行われているが、このシステムは並列度に関して限界がある。一方、トーラス型システムは、木型システムと同様、高並列度を実現するのに適した構造をもつ。提案するアルゴリズムでは、部分問題を各プロセッサに均一に配分するための負荷分散手法が工夫されている。そして、ナップサック問題を用いた場合のシミュレーション結果から、本論文で提案するアルゴリズムの加速指数が、木型システムを改良したＤＯＮシステム（二重結合木型システム）に対する既存のアルゴリズムの加速指数より大きくなることが確かめられる。また、ＤＯＮシステムに対するアルゴリズムと同様、問題の規模が大きくなるほど加速指数が増加し、しかも比較的計算時間を多く要す問題では加速指数がプロセッサの数より大きくなることが確かめられる。

A Parallel Branch-and-Bound Algorithm for a Torus Machine

Tsuyoshi KAWAGUCHI　　Tamotsu MAEDA　　Seiki KYAN

University of the Ryukyus

Abstract　In this paper we propose a parallel algorithm to execute branch-and-bound procedure on a torus machine with additional global links. The torus network is used when processors send subproblems to the respective adjacent processors in order to get a balanced work load. And global links are used for broadcasting the newly obtained temporary solution to all processors. Using the knapsack problem, the performance of the proposed algorithm is evaluated and is compared with the performance of a parallel brnch-and-bound algorithm for an improved tree machine.

第１章　まえがき

分枝限定法は、与えられた問題を部分問題に分割して解き、部分問題に対する最適解（局所最適解）の中で最良の解を全体の最適解とする手法である[1]。分枝限定法はほとんどすべての組合せ最適化問題に適用できる幅広い計算手法であるが、大規模な問題にこの手法を適用する場合には多くの計算時間を必要とする。このようなことから、最近、分枝限定法を並列計算機上で実行させるための並列分枝限定アルゴリズムに関する研究が行われてきている[2]～[6]。

並列計算機のアーキテクチャとしては様々なものが提案されているが[7]、並列分枝限定アルゴリズムに適したアーキテクチャはまだ確立されていない。文献（２）～（４）では共有メモリ型マルチプロセッサシステムに対するアルゴリズムが提案されている。また文献（５）では、木型システムを木の葉で連結した二重木型システム（ＤＯＮシステム）に対するアルゴリズムが提案されている。更に文献（６）では、プロセッサを環状に結合したシステム上でのアルゴリズムが報告されている。共有メモリ型システムはプロセッサ間の負荷分散を最も柔軟に行うことができるシステムであるが、プロセッサ間のメモリアクセス競合のため、その並列度は３０程度が限界である。また、木型システムやＤＯＮシステムは並列度を上げるためには優れているが、次のような問題点が指摘されている[5]。これらのシステムでは、各プロセッサは、親プロセッサ上の部分問題の分枝の結果得られる部分問題を受け取り、これを解く。従って、親プロセッサ上の部分問題に対して分枝が行われない場合には、子プロセッサは遊休状態に落ち入り、この遊休状態は更に下位にあるプロセッサに次々と伝播して行く。

本論文では、トーラス型マルチプロセッサシステム上での並列分枝限定アルゴリズムを提案する。このシステムはプロセッサを格子状に結合するので、木型システムと同様、並列度を上げるために適している。従って、プロセッサ間の負荷分散手法が重要となるが、本論文で提案する並列分枝限定アルゴリズムでは次のような負荷分散手法を用いる。各プロセッサは、部分問題を蓄えるためのスタックを持ち、各繰り返しにおいてスタックの先頭の部分問題を解く。そして、分枝の結果生じる部分問題の半分を自分のスタックに挿入し、残りの半分を下位プロセッサに送る。この操作によって、同じ列に配置されたプロセッサ間での負荷分散がはかられる。また、各繰り返しにおいて、プロセッサは左右のプロセッサと部分問題の数の均一化を行う。この操作によって、同じ行に配置されたプロセッサ間での負荷分散がはかられる。

ナップサック問題を用いた場合のシミュレーション結果から、本論文で提案したアリゴリズムの加速指数が、ＤＯＮシステムに対するアルゴリズム[5]の加速指数より大きくなることが確かめられる。また、ＤＯＮシステムに対するアルゴリズム同様、問題の規模が大きくなる程加速指数が増加し、しかも比較的時間を要す問題では、加速指数がプロセッサの数よりも大きくなるという結果が得られる。

第２章　トーラス型システム

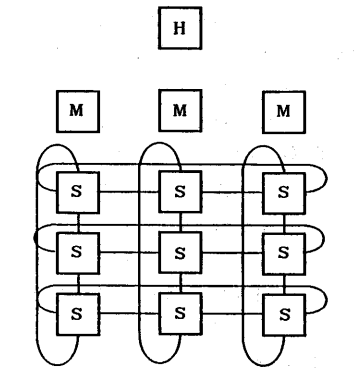
本論文で用いるトーラス型マルチプロセッサシステム（以後トーラス型システムと略す）は、ＳＬＡＶＥと呼ばれるm^2個のプロセッサ、ＭＡＳＴＥＲと呼ばれるｍ個のプロセッサ、及びホスト計算機（ＨＯＳＴ）から構成される。但し、ｍは任意の正の整数である。便宜上、m^2個のＳＬＡＶＥは図１に示すようにｍ行ｍ列の格子点上に配置されているものと考え、ｉ行ｊ列（$0 \leq i \leq m-1, 0 \leq j \leq m-1$）の位置にあるＳＬＡＶＥをＳＬＡＶＥ(ｉ,ｊ)で表す。但し、図１におけるＨ、Ｍ、Ｓは各々ＨＯＳＴ、ＭＡＳＴＥＲ、ＳＬＡＶＥを表す。更に、ＳＬＡＶＥ(ｉ,ｊ)に対して、ＳＬＡＶＥ((ｉ-１)mod ｍ,ｊ)、ＳＬＡＶＥ((ｉ+１)mod ｍ,ｊ)、ＳＬＡＶＥ(ｉ,(ｊ-１)mod ｍ)、ＳＬＡＶＥ(ｉ,(ｊ+１)mod ｍ)を、各々、上位ＳＬＡＶＥ、下位ＳＬＡＶＥ、左隣りのＳＬＡＶＥ、右隣りのＳＬＶＥと呼ぶ。以下に、トーラス型システムの詳細を記述する。

（１）　m^2個のＳＬＡＶＥは図１（ａ）に示すようなトーラス網によって結合されている。即ち、各ＳＬＡＶＥは上位、下位、左隣り、右隣りの４個のＳＬＡＶＥと結合されている。この結合網をローカル・リンクと呼ぶ。ローカル・リンクは、各ＳＬＡＶＥが自分に隣接する４個のＳＬＡＶＥと通信を行う場合に使用される。

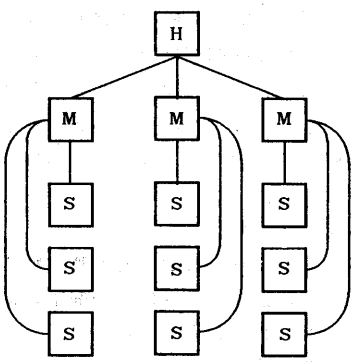
（２）　各列のｍ個のＳＬＡＶＥに対して、１個のＭＡＳＴＥＲが割り当てられる。そして図１（ｂ）に示すように、ＭＡＳＴＥＲは自分の配下

にあるm個のＳＬＡＶＥと結合され、ＨＯＳＴはすべてのＭＡＳＴＥＲと結合されている。この結合網をグローバル・リンク呼ぶ。グローバル・リンクは、ＳＬＡＶＥとＭＡＳＴＥＲ間、及びＭＡＳＴＥＲとＨＯＳＴ間の通信に使用される。例えば、ＨＯＳＴからすべてのＳＬＡＶＥへのブロードキャストは、まずＨＯＳＴがすべてのＭＡＳＴＥＲに信号を送り、次に各ＭＡＳＴＥＲが自分の配下にあるＳＬＡＶＥに信号を送ることによって実行される。

（３）　各ＳＬＡＶＥは、逐次処理用ランダムアクセス・マシン（ＲＡＭ）であり、ローカルメモリを持つ。



（ａ）　ローカル・リンク



（ｂ）　グローバル・リンク

S:SLAVE　M:MASTER　H:HOST

図１　トーラス型システム

第３章　並列分枝限定アルゴリズム

この章では、２章で述べたトーラス型システム上で動作する並列分枝限定アルゴリズムを提案する。まず３．１で分枝限定法について説明し、３．２で並列分枝限定アルゴリズムを記述する。

３．１　分枝限定法

文献（１）に示された０－１ナップサック問題の例を用いて分枝限定法を説明する。入力として次のような０－１ナップサック問題が与えられたとする。この問題をP_0で表す。

$$P_0 \quad : -3X_1-4X_2-X_3-2X_4 \rightarrow \text{最小}$$
$$2X_1+3X_2+X_3+3X_4 \leqq 4$$
$$X_1, X_2, X_3, X_4 = 0 \text{又は} 1$$

P_0に対して、例えば、$X_2=0$，$X_2=1$と固定することによって、２つの部分問題P_1，P_2が得られる。

このように、問題を互いに排反な部分問題の集合に分割する操作を分枝と呼ぶ。一般に部分問題P_iを分枝して部分問題P_{i1}，…，P_{ik}が得られるならば、これらの問題の間には

$$f(P_i)=\min(f(P_{i1}),\cdots,f(P_{ik}))$$

の関係が成り立つ。但し、$f(\cdot)$は対応する問題の最適解値を表す。従って、部分問題に対する最適解の中で最良の解を全体の最適解（大域的最適解）とすればよい。

分枝限定法は、分枝操作に加えて、枝刈りと呼ばれる操作を用いる。これは、大域的最適解が存在する可能性のない部分問題を、実際に解く前に削除する操作である。枝刈りを行うために、各部分問題P_iに対して下界値$g(P_i)$が計算される。$g(\cdot)$が下界値関数であるためには、各P_iに対して$g(P_i) \leqq f(P_i)$が成り立ち、しかも、P_iとP_iを分枝して得られるすべての部分問題P_kの間に$g(P_k) \geqq g(P_i)$の関係が成り立てばよい。例えば、０-１ナップサック問題に対しては「$X_i=0$又は１」の条件を、「$0 \leqq X_i \leqq 1$」に緩和して得られる線形計画問題の最適解値が下界値として用いられる。前述の例を更に続け、P_1に対して$X_4=0$，$X_4=1$と固定して得られる部分問題をP_3，P_4とすると$g(P_3)$を求める過程で整数解が得られる。つまり、P_3に対して「$X_1=0$又は１」や「$X_3=0$又は１」の分枝を行わなくても、P_3の最適解$f(P_3)$が得られたことになる。更にP_4に対しては$g(P_4)=-7/2 \geqq f(P_3)=-4$であり、$f(P_4)$を求めても$f(P_4)$は大域的最適解には成り得ないことがわかる。そこで、P_4は枝刈りされる。分枝限定法の計算途中において、それまでに見つかった可能解の中で最良の解を暫定解と呼ぶ。上記のP_4の例のように、暫定解値以上の下界値を持つ部分問題が枝刈りされる。

３．２　並列分枝限定アルゴリズム

３.２.１　概要

各ＳＬＡＶＥはスタックを持ち、スタックの中に活性な部分問題（分枝や枝刈りが適用されずにいる部分問題）を記憶する。各ＳＬＡＶＥは、スタックが空でないならば、スタックの先頭の部分問題P_iを取り出し下界値を計算する。この結果P_iが分枝され新たな部分問題が生成されるならば、その半分を自分自身のスタックに入れ、残りの半分を下位ＳＬＡＶＥに送る。また、各ＳＬＡＶＥは、自分が持つ部分問題の数が右隣りのＳＬＡＶＥの部分問題の数より２以上多いとき、部分問題の１個を右隣りのＳＬＡＶＥに送る。この方法によって、各ＳＬＡＶＥが持つ部分問題の数の均一化をはかる。

更に暫定解Ｚが更新される場合のブロードキャストは、次の方法で行われる。ＳＬＡＶＥは暫定解Ｚを記憶しており、各繰り返しにおいてこの値をＭＡＳＴＥＲに送る。ＭＡＳＴＥＲは送られてくるＺの中の最小値をＨＯＳＴに送り、ＨＯＳＴは送られくるＺの中の最小値をＭＡＳＴＥＲ経由ですべてのＳＬＡＶＥに送る。

３.２.２　アルゴリズムの詳細

各プロセッサの状態を次のように定義する。

実行状態：処理を実行中の状態

ｗａｉｔ状態：他のプロセッサから信号又はデータを待っている状態

停止状態：動作を停止している状態

停止状態からｗａｉｔ状態への移行は起動信号によって行われ、ｗａｉｔ状態から停止状態への移行はstop信号によって行われる。また、各プロセッサは、他のプロセッサから信号が送られてくるとｗａｉｔ状態から実行状態に移り、その処理が終了した時点でｗａｉｔ状態に戻る。

また、上記の起動信号、stop信号以外に次の信号が用いられる。

repeat信号：ＨＯＳＴがＭＡＳＴＥＲ経由で、ＳＬＡＶＥを実行状態に移す信号

idle信号：ＳＬＡＶＥがＭＡＳＴＥＲ経由でＨＯＳＴに対して、自分のスタックが空であることを伝える信号

busy信号：ＳＬＡＶＥがＭＡＳＴＥＲ経由でＨＯＳＴに対して、自分のスタックが空でないことを伝える信号

empty信号：各ＳＬＡＶＥが右隣り及び下位のＳＬＡＶＥに対して、伝送する部分問題が存在しないことを伝える信号

{起動ルーチン}

ＨＯＳＴが、次の（１）～（４）の順に処理を行う。

（１）起動信号をＭＡＳＴＥＲ経由ですべてのＳＬＡＶＥに送る（この信号によって、ＭＡＳＴＥＲやＳＬＡＶＥは起動されｗａｉｔ状態になる）。

（２）下界値計算のためのプログラム、入力データ（ナップサック問題の場合には、目的関数の係数と制約式の係数）及び暫定解Ｚ＝∞を、ＭＡＳＴＥＲ経由ですべてのＳＬＡＶＥにロードする。

（３）ＭＡＳＴＥＲ経由でＳＬＡＶＥ(０，０)に部分問題P_0をロードする。

（４）ＭＡＳＴＥＲ経由ですべてのＳＬＡＶＥにrepeat信号を送る。

{繰り返しルーチンと停止のための処理}

ＳＬＡＶＥ、ＭＡＳＴＥＲ、ＨＯＳＴごとに記述する。

・ＳＬＡＶＥの動作

ＭＡＳＴＥＲからのrepeat信号を受信すると、左右のＳＬＡＶＥ間での部分問題数の均一化、スタックの先頭の部分問題に対する下界値の計算と分枝、上下のＳＬＡＶＥ間での部分問題の移動の処理を連続して行う。この一連の処理を１回の繰り返しと呼ぶ。なお、ＳＬＡＶＥの動作の詳細を図２に示す。

・ＭＡＳＴＥＲの動作

（１）ＨＯＳＴからrepeat信号と暫定解Ｚが送られてくれば、これを自分の配下にあるすべてのＳＬＡＶＥに送る。

（２）ＨＯＳＴからstop信号が送られてくれば、これを自分の配下にあるすべてのＳＬＡＶＥに送り、自分自身の動作を停止する。

（３）ＳＬＡＶＥからidle信号又はbusy信号が送られてくれば、この数をカウントする（但し、(１)でＳＬＡＶＥにrepeat信号を送るたびにカウント値を０にクリアしておく）。そして、このカウント値がｍ（自分の配下にあるＳＬＡＶＥの数）に等しくなった時点で、２つの場合ごとに処理を行う。

```
procedure SLAVE
begin
  if MASTARからstop信号が送られてくる  then
     動作を停止する;
  if MASTERからrepeat信号とZが送られてくる
     then begin
     自分のもつ暫定解Z(S)をZに更新する;
     if （自分のもつ部分問題の数）≧（右隣り
          のSLAVEがもつ部分問題の数）+2 then
          部分問題の１個を右隣りのSLAVEへ送る;
     else
          empty信号を右隣りのSLAVEへ送る;
     左隣りのSLAVEから部分問題又はempty信号が
     送られてくるの待ち、もし部分問題が送られ
     てくればこれをスタックへ挿入する;
     if スタックが空でない then begin
        スタックの先頭の部分問題Piを取り出す;
        Piの下界値g(Pi)を計算する;
        if 最適解f(Pi)が得られる then begin
           if Z(S)>f(Pi) then
              Z(S)をf(Pi)に更新する;
           下位SLAVEにempty信号を送る;
           end
        else
           if g(Pi)<Z(S) then begin
              Piを部分問題Pi1,…,Pikに分割する;
              Pi1,…,Pi⌈k/2⌉ をスタックの先頭に
              挿入;
              Pi⌈k/2⌉+1,…,Pikを下位SLAVEへ送る;
           end
     end {if}
     上位SLAVEから部分問題又はempty信号が送ら
     れてくるの待ち、部分問題が送られてくれば
     これをスタックへ挿入する;
     if スタックが空である then
        idle信号とZ(S)をMASTERへ送る;
     else
        busy信号とZ(S)をMASTERへ送る;
  end {if}
end {SLAVE}
```

図２．ＳＡＬＶＥの動作

ｃａｓｅ１：すべてのＳＬＡＶＥからの信号がidle信号であった場合

このときＨＯＳＴにidle信号を送る。

ｃａｓｅ２：少なくとも一つのＳＬＡＶＥからの信号がbusy信号であった場合

busy信号と、ＳＬＡＶＥから送られてきた暫定解Ｚの中での最小値をＨＯＳＴに送る。

・ＨＯＳＴの動作

ＭＡＳＴＥＲにrepeat信号を送った後、新たにＭＡＳＴＥＲから送られてくるidle信号又はbusy信号の数をカウントしておく。そして、この数がｍ（ＭＡＳＴＥＲの数）に等しくなった時点で、２つの場合ごとに以下の処理を行う。

ｃａｓｅ１：すべてのＭＡＳＴＥＲからの信号がidle信号であった場合

このときすべてのＭＡＳＴＥＲにstop信号を送る。

ｃａｓｅ２：少なくとも一つのＭＡＳＴＥＲからの信号がbusy信号であった場合

repeat信号とＭＡＳＴＥＲから送られてきた暫定解Ｚの中での最小値をすべてのＭＡＳＴＥＲに送る。

３．３　計算時間の評価

ここでは、３．２で述べたアルゴリズムの１回の繰り返しに要す時間を評価する。便宜上、各ＳＬＡＶＥに対して番号ｉ（$1 \leq i \leq m^2$）が与えられているものとし、番号ｉをもつＳＬＡＶＥの上位ＳＬＡＶＥの番号をＵ(ｉ)で表す。

番号ｉ（$1 \leq i \leq m^2$）を持つＳＬＡＶＥが、ＭＡＳＴＥＲからのrepeat信号を受信した後ＭＡＳＴＥＲへbusy信号（又はidle信号）を送信するまでの時間をｔ(ｉ)で表す。更に、

$t_r(i)$:部分問題の１個を右隣りのＳＬＡＶＥに送るのに要す時間

$t_l(i)$:左隣りのＳＬＡＶＥから送られてきた部分問題をバッファからスタックへ移すのに要す時間

$t_c(i)$:スタックの先頭の部分問題を取り出し、この部分問題に対して下界値計算と分枝操作を実行するのに要す時間

$t_d(i)$:分枝して生成された部分問題の半分を下位ＳＬＡＶＥへ送るのに要す時間

$t_u(i)$:上位ＳＬＡＶＥから送られてきた部分問題をバッファからスタックへ移すのに要す時間

とする。すべての部分問題にに対して部分問題を表現するのに必要なバイト数が等しい場合には、$t_r(i)$、$t_l(i)$、$t_d(i)$、$t_u(i)$の値はｉに依らず一定の値をもつ。そこで、これらの値をt_r、t_l、t_d、t_uで表すと次式が成り立つ。

$$t(i)=t_r+t_d+\max(p(i),p(U(i))) \quad (1)$$

但し

$$p(i)=t_l+t_c(i)+t_u \quad (2)$$

それゆえ、

t_{q1}：ＨＯＳＴがすべてのＳＬＡＶＥに対してrepeat信号と暫定解Ｚをブロードキャス

トするのに要す時間

t_{q2}：すべてのＳＬＡＶＥが同時にbusy信号（又はidle信号）と暫定解ＺをＨＯＳＴへ送ってから、ＨＯＳＴがＺの最小値を求めるまでの時間

とするとアルゴリズムの１回の繰り返し時間Ｔは

$$T_\theta + t_r + t_d + t_{q1} \leqq T_\theta + t_r + t_d + t_{q1} + t_{q2} \quad (3)$$

を満たす。但し

$$T_\theta = \max\{p(i) \mid 1 \leqq i \leqq m_2\} \quad (4)$$

であり、$p(i)$は式（２）に示された値である。更に

ｆ：部分問題を表現するのに必要なバイト数

ｋ：１回の分枝操作で生成される部分問題の数

t_θ：隣接するプロセッサ間で１バイトのデータ伝送に要す時間

とすると、式（２）、（３）に示された各値は次式で与えられる。

$$t_r = f \cdot t_\theta \ , \ t_d = \lceil k/2 \rceil \cdot f \cdot t_\theta$$
$$t_{q1} = t_{q2} = 2m t_\theta \quad (5)$$

第４章　アルゴリズムの性能評価

本章では、第３章で述べたトーラス型システムに対する並列分枝限定アルゴリズムの性能を、文献（５）のＤＯＮシステムに対するアルゴリズムの性能とシミュレーションにより比較した結果を示す。アルゴリズムを適用した問題は、３．１で述べた０－１ナップサック問題である。また、アルゴリズムの性能を表す尺度としては

$$Rs = \frac{\text{逐次分枝限定アルゴリズムの実行時間}}{\text{並列分枝限定アルゴリズムの実行時間}} \quad (6)$$

で定義される加速指数Ｒｓを用いた。シミュレーション結果を示す前に、ＤＯＮシステムについて簡単に説明しておく。ＤＯＮシステムは、Ｄ－Ｎｅｔ及びＭ－Ｎｅｔと呼ばれる２個の木型システムを各々の葉で連結したシステムである。Ｄ－Ｎｅｔを構成するプロセッサはDECOMPOSERと呼ばれ、部分問題に対する下界値計算や分枝を実行する。一方、Ｍ－Ｎｅｔを構成するプロセッサはMERGERと呼ばれ、暫定解値Ｚより小さい下界値をも部分問題のみを選び、ＳＣ（SYSTEM CONTROLER）に伝える働きをもつ。そして、ＳＣはMERGERから送られてくる部分問題をスタックへ挿入するとともに、スタックの先頭の部分問題をＤ－Ｎｅｔの根に送る働きをもつ。

以下に、シミュレーションに用いた具体的な処理について説明する。

（１）　トーラス型システムのアルゴリズムに対しては、ｊ（≧１）回目の繰り返しにおける実行時間を式（６）によって与え（これを$T_\theta(j)$で表す）、

$$\sum_{j=1}^{J} T_\theta(j) \quad （Ｊ：繰り返し回数）$$

によって実行時間を与えた。またＤＯＮシステムのアルゴリズムに対しても、すべての通信時間は０と仮定した。

（２）　トーラス型システムにおけるＳＬＡＶＥ数m^2を$m^2 = 16$と設定した。従って、ＨＯＳＴも含めた全プロセッサ数は２１（$=m^2+m+1$）である。一方、ＤＯＮシステムとしては深さｄ＝３のＤＯＮシステムを用いた。このシステムの全プロセッサ数は３１（$=2^{d+2}-1$）であり、トーラス型システムのＳＬＡＶＥに対応する働きをもつDECOMPOSERの数は１５（$=2^{d+1}-1$）である。なお、$m^2 = 16$及びｄ＝３と設定した理由は、同じ働きをもつＳＬＡＶＥとDECOMPOSERの数をほぼ等しくするためである。

（３）　（２）で述べたように、実験で用いたトーラス型システムとＤＯＮシステムのプロセッサ数は異なる。従って、式（６）の加速指数Ｒｓによって両者を単純に比較することはできない。そこで、

N_P：全プロセッサ数（トーラス型システムではm^2+m+1、ＤＯＮシステムでは$2^{d+2}-1$）

N_{SD}：トーラス型システムに対してはＳＬＡＶＥ数m^2を表し、ＤＯＮシステムに対してはDECOMPOSER数$2^{d+1}-1$を表す値

とし、Rs/N_P及びRs/N_{SD}によって両者を比較する。

なお、Rs/N_{SD}値の比較は次のような意味をもつ。第２章で述べたトーラス型システムにおけるＭＡＳＴＥＲは、単に暫定解Ｚの更新のための処理と同期のための処理を行う。同様にＤＯＮシステムにおけるMERGERも、単にＺより小さい部分問題を選びSYSTEM CONTROLERに伝える処理を行う。従ってＭＡＳＴＥＲやMERGERは、部分問題の下界値計算や分枝などの高度な演算を必要とするＳＬＡＶＥやDECOMPOSERよりも、低価格のプロセッサで構成できる。

まず、乱数によって部分問題の最適値や下界値を与えるシミュレーションモデル[1]、[5]を用いた場合の結果を図３に示す。図３の横軸はナップサック問題の変数の個数ｎであり、縦軸はＲｓ／N_P及びＲｓ／N_{SD}である。なお、すべての測定点はそれぞれ２０個の問題に対する平均値である。図３の実験では実際にはナップサック問題を解かないので、すべての部分問題に対して下界値や分枝を行うのに要す時間は等しいと仮定されている。

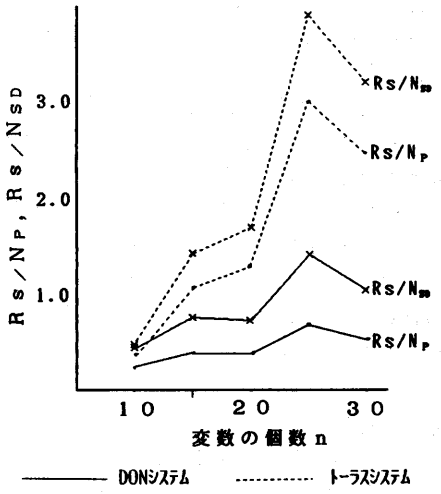


図３　シミュレーションモデルを用いた場合の結果

また、実際にナップサック問題を解いた場合の結果を図４（ａ）、（ｂ）に示す。図４におけるすべての測定点はそれぞれ２０個の問題に対する平均値である。図４の実験においては、ナップサック問題を規定する目的関数の係数c_i、制約式の係数a_i及び制約式の右辺の値ｂは一様分布に従うものとし乱数によって与えた。なお、これらの値を与える方法として次の（Ａ）、（Ｂ）２通りの方法を用いた。そして、図４（ａ）は（Ａ）の方法を用いた場合の結果であり、図４（ｂ）は（Ｂ）の方法を用いた場合の結果である。

（Ａ）　c_i、a_i、ｂはそれぞれ［１，１００］、［１００，１００＋ｎ／２］、［１００，Σa_i］の整数値上の一様分布に従う。

（Ｂ）　c_i、a_i、ｂはそれぞれ［１，１００］、［１００，１０５］、［１００×ｎ／５，（Σa_i）／２］の整数値上の一様分布に従う。

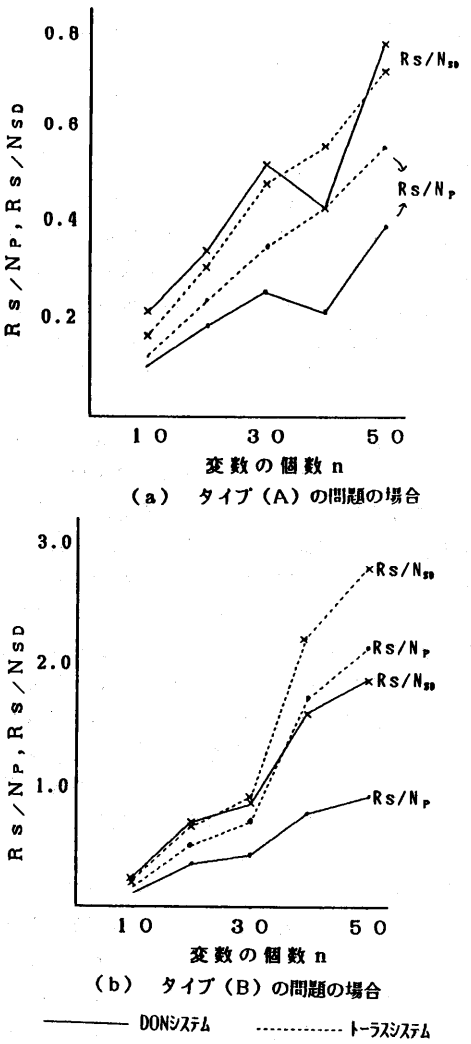


図４　ナップサック問題に対するシミュレーション結果

上記の（Ａ）は最適解が比較的容易に求まる問題例を生成し、（Ｂ）は最適解を求めることが比較的難しい問題例を生成する。例えば、ｎ＝４０に対して逐次分枝限定アルゴリズムによって生成された部分問題の数は、（Ａ）の場合で平均３５５個、（Ｂ）の場合で平均１７８０個であった。

図３、図４から次のことがいえる。

（ｉ）　図３、図４（ａ）、（ｂ）のすべての実験に対して、本論文で提案したアルゴリズムのＲｓ／N_P値は文献（５）のアルゴリズムのＲｓ／N_P値より大きな値をもつ。このことは、プロセッサ数が等しいトーラス型システムとＤＯＮシステムを用いたとき、本論文のアルゴリズムが文献（

5）のアルゴリズムよりも大きな加速指数を与えることを意味する。

（ii） Rs/N_{SD}に関しても、図4（a）の実験以外は、本論文のアルゴリズムのRs/N_{SD}値が文献（5）のアルゴリズムのRs/N_{SD}値より大きな値をもつ。前述したように、図4（a）の実験に用いられた問題は最適解を求めることが比較的容易な問題であり、（b）の実験に用いられた問題は最適解を求めることが比較的難しい問題である。従って図4の結果は、多くの計算時間を要す問題ほど本論文のアルゴリズムが有効となることを示している。

（iii） 本論文のアルゴリズムと文献（5）のアルゴリズムの両方とも、問題の規模が増加するほど加速指数が増加し、しかも比較的計算時間をを要す問題では、加速指数がプロセッサの数よりも大きくなる（この現象は加速異常と呼ばれる）。

3．2で述べた負荷分散手法によって部分問題が全体のＳＬＡＶＥに行き渡るには、少なくともm回の繰り返しが必要である。図4（a）に示したように、比較的容易に解ける問題に対して本論文のアルゴリズムの加速指数が大きくならないのは、このためと考えられる。しかし、一度部分問題が全体のＳＬＡＶＥに行き渡ると負荷分散手法は有効に働くことになる。

本論文のアルゴリズムと文献（5）のアルゴリズムの一つの大きな相違点は、前者が部分問題を各ＳＬＡＶＥのスタックに蓄えるのに対して、後者は部分問題のすべてを共通のスタックに蓄える点である。第1章で述べたように、文献（5）のアルゴリズムでは、あるプロセッサ上の部分問題が終端されるとそれより下位のプロセッサが次々と遊休状態に落ち入ることになる。この問題に対する対策として、ＤＯＮシステムに対しても、各プロセッサが部分問題の一部を自分のスタックに蓄えておく方法が考えられる。しかし例えば、各部分問題から2個の部分問題が生成されるような場合（実際分枝限定法の多くの適用例でそのような分枝規則が用いられる）、プロセッサが自分に1個の部分問題を残すと、左の子又は右の子のいずれかがやはり遊休状態になる。このように木型システムやＤＯＮシステムでは、各プロセッサが部分問題を自分のスタックに蓄える場合でも、各プロセッサ間で部分問題の数の均一化をはかる操作が難しくなる。

第5章 むすび

トーラス型マルチプロセッサシステムは、木型マルチプロセッサシステムと同様、高並列度を実現するのに適した並列計算機アーキテクチャと考えられている。本論文では、組合せ最適化問題に対する最も一般的な計算手法である分枝限定法を、トーラス型マルチプロセッサシステム上で実行させるためのアルゴリズムを提案した。そしてシミュレーション結果から、提案したアルゴリズムの加速指数が、木型システムを改良したＤＯＮシステムに対する既存のアルゴリズム[5]の加速指数より大きくなることが確かめられた。

分枝限定法は基本的には木探索（トリー・サーチ）であるので、本論文のアルゴリズムはPrologの並列処理などにも応用できる。しかしProlog処理の場合には、分枝して得られる各ノードが根から自分に至る道上のすべての履歴を記憶しておかなければならない。従って、プロセッサ間でのノードの移動をできるだけ少なくするような負荷分散手法が必要になる。

［参考文献］

（1）茨木俊秀：”組合せ最適化ー分枝限定法を中心として（講座数理計画法8）”，産業図書（昭58）．

（2）今井，吉田，福村：”分枝限定法アルゴリズムの並列化とその評価”，信学論，J62-D,pp.403-410（昭54－6）

（3）B.W.Wah and Y.W.E.Ma：”NANIP-a multicomputer architecture for soving combinatorial extremum-search problems”，IEEE Trans. Computers,C-33,5,pp.377-390(1984)

（4）伊藤，笠原：”最適化マルチプロセッサスケジューリングアルゴリズムの並列処理手法”，昭62前期情処全大，4Q－5．

（5）M.Imai：”A double-tree-structured multicomputer system and its application to combinatorial problems”，Trans.IECE Japan,E69,9,pp.1002-1010(1986-9)

（6）O.Vornbeger：”Implementing branch-and-bound in a ring of processors”，CONPAR 86, Proceedings of Conference on Algorithms and Hardware for Parallel Processing,Aachen,1986, pp.157-164.

（7）富田眞治：”並列計算機構成論”，昭晃堂（昭61）．