メッシュ結合並列計算機用パーティショニングアルゴリズムの時分割処理化

須崎有康† 田沼均† 平野聡†
一杉裕志† 塚本享治†

メッシュ結合並列計算機上でタスク割当てを行うパーティショニングアルゴリズムを時分割処理（TSS）に拡張した．この拡張により，パーティショニングアルゴリズムで問題となった応答を改善したばかりでなく，タスク割当て効率も向上させることができた．本TSSでは仮想並列計算機を用意し，これらの仮想並列計算機上にパーティショニングアルゴリズムによりタスク割当てを行う．各仮想並列計算機はラウンドロビンにより，実並列計算機で処理を進める．また，ある仮想並列計算機でタスクが占有している領域を別の仮想並列計算機では未使用の場合，そのタスクが複数の仮想並列計算機に存在することによりプロセッサ利用率の向上を計る．

Transformation of Partitioning Algorithms into Time Sharing Systems on Mesh-connected Parallel Computers

KUNIYASU SUZAKI,† HITOSHI TANUMA,† SATOSHI HIRANO,†
YUUJI ICHISUGI† and MICHIHARU TSUKAMOTO†

This paper presents a Time Sharing System which uses a partitioning algorithm for mesh-connected parallel computers. The TSS can shorten the response of tasks and increase the utilization of computers. Partitioning algorithms partition a mesh of processors into sub-meshs so that incoming tasks fit within the sub-meshes. The TSS has virtual parallel computers. The virtual parallel computers are executed by a real parallel computer alternately. Each virtual parallel computer is partitioned into sub-meshes by a partitioning algorithm. If the corresponding areas of a sub-mesh of a task on a virtual parallel computer are free on other virtual parallel computers, the task can sit on these virtual parallel computers in order to attain better processor utilization.

1. はじめに

現在，並列計算機の効率的利用を目指してパーティショニングアルゴリズムの研究が行われている．パーティショニングアルゴリズムは，投入されるタスクが要求するプロセッサ数や形状に合わせて，FCFS（First Come First Serve）で物理プロセッサに効率良く割り当てるアルゴリズムである．これらアルゴリズムによって複数のタスクを同時に実行できるようになり，また，全体のプロセッサのうち稼働しているプロセッサの比率（プロセッサ利用率）を高めることができる．

しかし，パーティショニングアルゴリズムで処理しても，プロセッサ空間の大部分を占有するタスクが到着すると，その前後のタスクを組み合わせてプロセッサ空間に割り当てることができない閉塞状態[1]になる．**図1**を例にとると，タスク1，2，4のみならばパーティショニングアルゴリズムを使って3つのタスクを同時に実行することができる．しかしタスク3が間に入るため，タスク1，2の処理後（図1左下），タスク3を処理し（図1中下），タスク4を処理する（図1右下）ことになる．このような状況ではプロセッサの利用率が低下し，効率的利用ができない．さらに，個々のタスクの応答も閉塞状態のために悪化する．

本論文では，このような状態を回避する手法としてパーティショニングアルゴリズムを拡張した時分割処理（TSS: Time Sharing System）を提案する．提案するTSSでは，実並列計算機と同一形状の仮想並列計算機を用意し，タスクの割当ては各仮想並列計算機にパーティショニングアルゴリズムを用いて行う．1台の仮想並列計算機に割り当てられなかったタスクは，別の仮想並列計算機を新たに用意し，そこに割り当て

† 電子技術総合研究所
Electrotechnical Laboratory

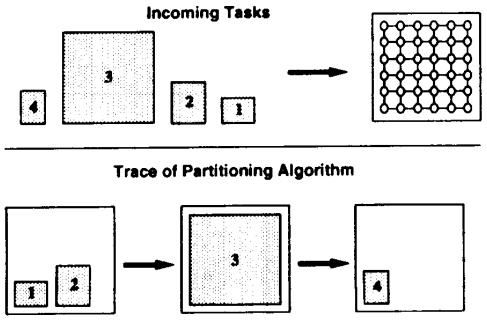

図1 閉塞状態になるタスクの投入
Fig. 1 Tasks which cause blockade situation.

る．これらの仮想並列計算機は実並列計算機で一定時間ごとに処理を進められる．このTSS化によって，1つの仮想並列計算機でタスク1，2，4を処理し，もう1つの仮想並列計算機でタスク3を処理することで閉塞状態を回避することができる．

現在TSSを採用している商用並列計算機としてCM5[2]がある．CM5ではシステムの立ち上げ時にパーティションした領域個々に対してTSSを行う．しかし，立ち上げ時に決められたパーティションを変えることはできないため，立ち上げ時にパーティションした数以上のタスクを同時実行することができない．このため長期運用では，パーティションの大きさとタスクの大きさの不整合が生じ，プロセッサの効率的な利用が望めない．

本論文では二次元メッシュ結合並列計算機を対象とし，パーティショニングアルゴリズム2D Buddy[3),4)]とAdaptive Scan[5)]を拡張したTSSを提案する．以下2章で本TSSが想定するTSSの環境について概観する．3章ではパーティショニングアルゴリズムを紹介し，4章でTSSへの拡張方法を説明する．5章では提案するTSSの実行効率をシミュレーション結果で示す．6章で関連研究との比較を行い，7章で本方式の課題を議論する．8章で結論を述べる．

2. 想定するTSSの環境

提案するTSSは二次元メッシュ結合の並列計算機を対象とする．並列計算機はサブメッシュを動的に切り出すことができ，タスクはサブメッシュに割り当てられる．タスク間の通信やリモートメモリアクセスはサブメッシュの外部に対して干渉せず，また他のタスクの通信，リモートメモリアクセスによる干渉もない性質（閉パーティショニング）を有する．タスクのX方向とY方向を転置した形状のサブメッシュ上にタスクの割当ても可能であり，効率は変わらないものとする．また並列計算機は一定時間単位で全空間のプロセスを切り替えられる機構を有するものとする．切替えに要する時間については7章で議論する．

割り当てられたタスクのプログラムコードやデータは，各プロセッサが有するメモリ上に納められるものとする．この仮定により，TSS化で1つのプロセッサに複数のタスクが割り当てられてもメモリが不足することがない．メモリ不足が生じた場合の影響は7章で議論する．

投入されるタスクは並列計算機が提供する物理形状を越えない任意の長方形のプロセッサを要求するものとする．このタスクは終了するまで最初に要求したプロセッサ以上のプロセッサを要求したり，解放したりすることはしない．

3. パーティショニングアルゴリズム

メッシュ結合並列計算機を対象とするパーティショニングアルゴリズムにはFrame Slide[6)]，First Fit[7)]，Best Fit[7)]，Adaptive Scan[5)]，2D Buddy[3),4)]などがある．これらのうち本論文では，時分割処理の適用性を示すため2つのパーティショニングアルゴリズム2D BuddyとAdaptive Sacnに対してTSS化を行った．2D Buddyはいままでの研究によりプロセッサ利用率が悪いことが示されているが，その性質がよく解析されているため比較対象として採用し，TSS化による改善を調べる．また，2D Buddyはタスク管理データ構造が単純なため，4.2節で述べるmultipleタスクの実現が効率良く行える特色を持つ．Adaptive Sacnは単体でもプロセッサ利用率が高いことが知られているが，TSS化によりどの程度効率向上が行えるか調べる．

3.1 2D Buddy

2D Buddy (2DB)[3),4)]では，メッシュ結合の一辺のプロセッサ数が2の巾乗の正方形であることを仮定している．2DBではサブメッシュの形状を並列計算機より小さい2の巾乗の正方形とする．この正方形のサブメッシュは，メッシュ結合しているプロセッサを4分割（縦方向2分割，横方向2分割）を繰り返して得られる正方形の位置に割り当てられる．投入された任意の長方形のタスクは，長辺lが$2^{n-1} < l \leq 2^n$であるサブメッシュに割り当てられる．

サブメッシュの空き状態は個々の大きさごとにFBL (Free Block List) で管理し，該当する大きさのFBLが空の場合はより大きいサブメッシュを分割してタスクに割り当てる．FBLよりタスクが管理されている状態を図2に示す．

2DBによるサブメッシュの割当てと解放の手続きを

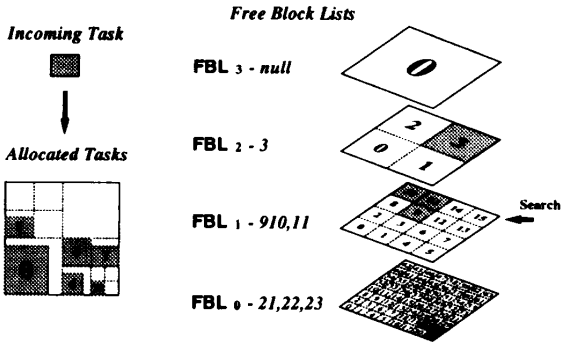

図2 2D Buddy の管理
Fig. 2 Management by 2D Buddy.

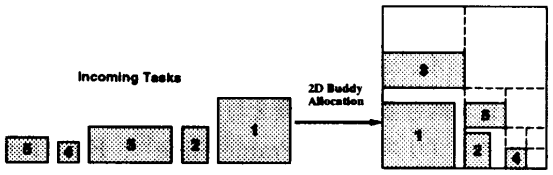

図3 2D Buddy によるサブメッシュの割当て
Fig. 3 Submesh allocation by 2D Buddy.

付録に示す．この手続きに従った割当ての一例を図3に示す．投入されるタスクの大きさを (w,h) とすると，一辺が $2^{\max(\lceil \log w \rceil, \lceil \log h \rceil)}$ の正方形がサブメッシュとしてとられる．このうちタスクが使用するのは $w \times h$ であるので，残りの $2^{\max(\lceil \log w \rceil, \lceil \log h \rceil)} - w \times h$ は未使用であるが他のサブメッシュとして使用することはできない．この領域を内部フラグメンテーションと呼ぶ．割当てが行われていない領域を外部フラグメンテーションと呼ぶ．

3.2 Adaptive Scan

Adaptive Scan (AS)[5]では投入されたタスクと同じ長方形のサブメッシュを，メッシュの下端 $(0,0)$ からX方向にサーチし，他のサブメッシュと重ならない最初の位置に割り当てる．見つからなかった場合は，Y方向に1つプロセッサ番号を進め繰り返す．全領域をサーチしても適するサブメッシュが見つからないときは，タスクのX方向の大きさとY方向の大きさを転置してもう一度サーチする．

ASのサブメッシュのサーチは，1つの要素が1台のプロセッサを表すビットマップを用いる．ここではサブメッシュを割り当てられない領域を示すreject setとcoverage setがある．メッシュの大きさを (M,N)，投入されるタスクの大きさを (w,h) とすると，reject setは，投入されたタスクのサブメッシュの左下端を割り当てることができないX方向の領域（$M-w+1$ から $M-1$）とY方向の領域（$N-h+1$ から $N-1$）

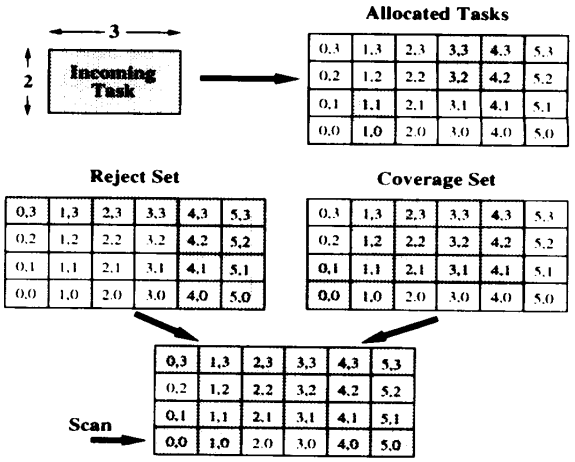

図4 Adaptive Scan の管理
Fig. 4 Management by Adaptive Scan.

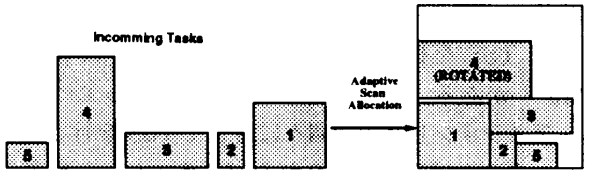

図5 Adaptive Scan によるサブメッシュの割当て
Fig. 5 Submesh allocation by Adaptive Scan.

を示す．coverage setは，すでにサブメッシュ（左下端が (p,q)，占有する大きさが (s,t)）が割り当てられているために投入されたタスクのサブメッシュの左下端を割り当てられない領域（$(p-w+1, q-h+1)$ と $(p+s-1, q+t-1)$ を囲む長方形部分）を示す．ASのサーチでは，reject setとcoverage setをマージしたビットマップからreject setでなくcoverage setでもない部分を探す．図4にASにサブメッシュサーチの一例を示す．

ASによるサブメッシュの割当てと解放の手続きを付録に示す．この手続きに従った割当ての動作の一例を図5に示す．ASではタスクの長方形と同じ大きさの長方形のサブメッシュを割り当てるので，2DBで発生した内部フラグメンテーションは起こらない．

4. 並列計算機のTSS

逐次計算機のTSSではタスクに対し一定時間ごとに計算機資源を割り当てていたが，本論文で提案するTSSでは実並列計算機と同一の構成の仮想並列計算機を用意し，この仮想並列計算機を一定時間ごとに実並列計算機に割り当てることによって処理を進める．この仮想並列計算機を本論文ではスライスと呼ぶ．タスクはスライスにパーティショニングアルゴリズムによって割り当てられ，スライス数は投入されるタスク

の数によって変化する．スライスへの時間割当てはラウンドロビンで行われる．この状況を図**6** に示す．

並列計算機の TSS では，逐次計算機では問題にならなかったプロセッサ利用率の問題がある．1つのスライスにおいてはパーティショニングアルゴリズムによって利用率をあげることができる．さらに複数のスライス間ですでに割り当てられているタスクの領域が他のスライスで未使用の場合，そのタスクは複数のスライスに存在しプロセッサ利用率をあげることができる．本論文では1つのスライスにのみ存在するタスクを *single* タスク，他のスライスに存在することができるタスクを *multiple* タスクと呼ぶ．この状況を図6 に示す．

4.1 TSS アルゴリズム

本節では TSS を管理するためのデータ構造とアルゴリズムについて述べる．single と multiple のタスクの管理は図**7** に示すデータ構造で管理される．各スライスは，single と multipe のタスクを管理するリストを持ち，リストにはタスクの割り当てられている領域情報（サブメッシュの最小 X 座標と Y 座標，およびサブメッシュの X 方向の大きさと Y 方向の大きさ）が格納されている．multiple リストでは，さらに multiple となっているスライスの相互のスライス番号を保持する．この情報は図6 の点線で示されているように，2つのスライスにまたがるタスクは相互のスライス番号を持ち，3つのスライスにまたがるタスクはスライスが循環する番号を持つ．

本 TSS はラウンドロビンでスライスに時間を割り当てているので，応答を良くするにはスライス数を少なくする必要がある．スライスの削除はそのスライスに single タスクがなくなったときに行う．もし multiple タスクが存在すれば，その multiple タスクを持つスライスの multiple 情報を書き換え，削除される．この動作を multiple タスクの縮退と呼ぶ．縮退したタスクが1つのスライスのみで存在するタスクとなれば，その multiple タスクは single タスクに変わる．投入されたタスクと終了したタスクの管理手続きを次に示す．

タスク投入手続き

(1) スライスに対してを順次パーティショニングアルゴリズムでサブメッシュのサーチを行う．この際，各スライスには single タスクのみが存在するものとし，最初に見つかったスライスに対して割当てを行う．割当て可能なスライスが見つからなければ，ステップ(4)へ．

(2) 別のスライスにも同じサブメッシュが割り当てられれば multiple として登録，割り当てられなければ single として登録する．

(3) タスクを割り当てた領域に multiple タスクが存在していればその multiple タスクを縮退し，リターン．

(4) 新しいスライスを作成し，パーティショニングアルゴリズムでサブメッシュを割り当てる．

(5) 新しいスライスに他のスライスの single タスクが割り当てられるか探し，割り当てられたら multiple タスクにして割り当てる．新しいスライスに他のスライスの multiple タスクが割り当てられるか探し，割り当てられたら割り当て，

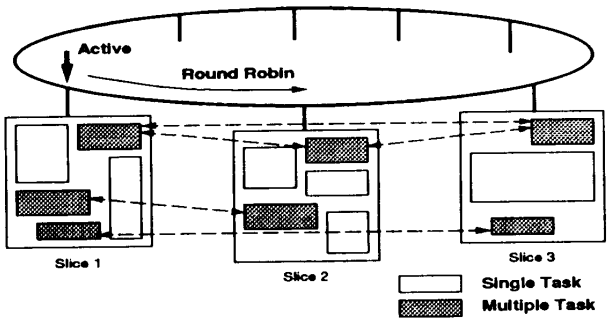

図6 スライスに基づくタスクの割当て
Fig. 6 Task allocation on slices.

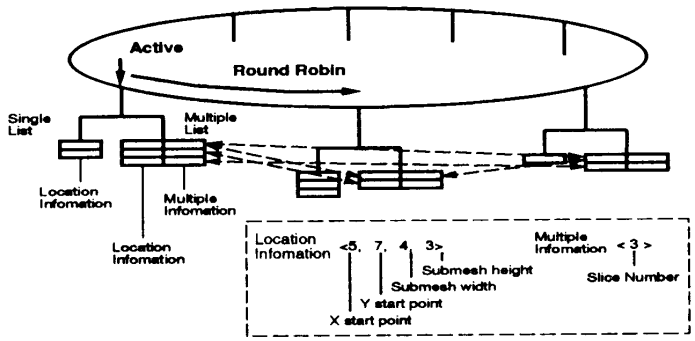

図7 スライス管理のデータ構造
Fig. 7 Data structure for slice management.

リターン.

タスクの解放手続き

(1) 終了したタスクが multiple なら，そのタスクを multiple リストから削除．ステップ(2)へ．終了したタスクが single であり，そのスライスから single タスクがなくなるなら，そのスライス上の multiple タスクを縮退し，スライスを削除．そうでないなら，タスクを single リストから削除する．ステップ(3)へ．

(2) 終了したタスクの multiple を持つスライスから，そのタスクを削除する．

(3) 割り当てられていたタスクがなくなったスライスに対して，他のスライスにある single タスクを multiple として割り当てられれば，multiple タスクにして割り当てる．次に multiple タスクを調べ，割り当てられれば割り当てる．

(4) single タスクを multiple に変えたスライスから single タスクがなくなれば，そのスライスは multiple タスクを縮退した後，スライスを削除して，リターン．

4.2 重なり判定

タスクを multiple として他のスライスに割り当てられるか割り当てられないかは，タスクのサブメッシュ領域が他のスライス上のタスクのサブメッシュと重ならないかの判定に基づく．サブメッシュの重なり判定は 2DB，AS それぞれの特徴を生かして次の手法で行っている．

2DB のサブメッシュの割当ては X 方向，Y 方向とも 2 の巾ごとの 4 分割に従うので，四分木による管理を行うことができる．重なり判定はこの四分木を検索すること実現できる．プロセッサ数を n としたときの検索オーダは $O(\log_4 n)$ である．

AS では，長方形のタスクを任意に投入するので 2DB の四分木のような効率的な管理ができない．しかし，長方形の重なり問題は計算幾何学の分野でよく研究されており，長方形の数が N，交差の数が I のとき，$O(N \log N + I)$ である効率的な判定アルゴリズム[8]がある．

4.3 TSS の動作

タスクの投入時，終了時のスライス上のタスクの独特な動作について説明する．

タスクが投入され multipe であった既存のタスクが single に変わる例を**図 8** に示す．TSS では，タスクのためのサブメッシュの割当てを single タスクのみが存在するものとして行う．次に割当てが成功の場合，その領域が multiple として使われていないか調べ，使われていたら multiple のタスクを削除する．図 8 の上段で New Task が mutiple タスク A" (' は multiple の個数を表す) の位置に割り当てられるため，mutiple タスク A" は削除される．multiple タスク A" はほかに multiple がないため，single に変化する (図 8 下段)．

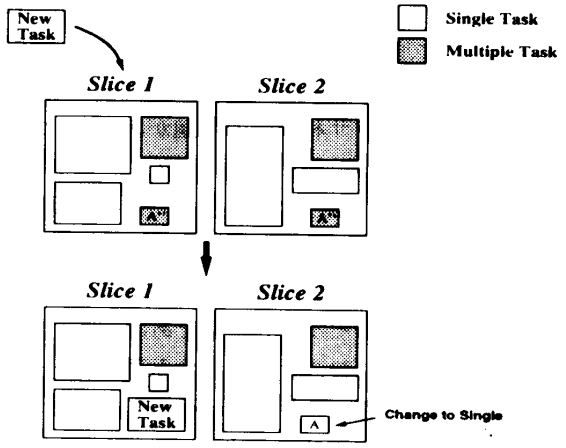

図 8 タスクの投入
Fig. 8 Task invoking.

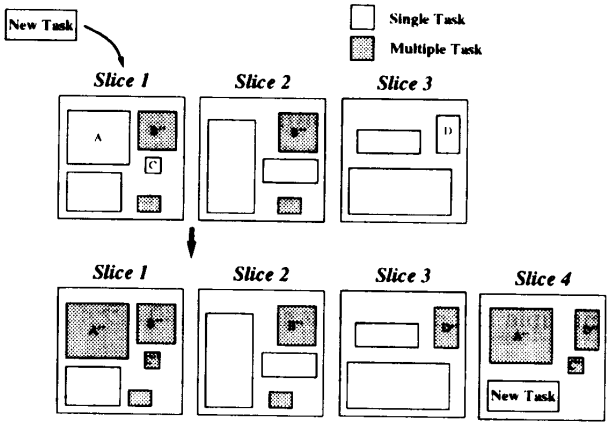

図 9 新しいスライスの作成
Fig. 9 New slice creation.

新しいスライスが作成されるときの動作を**図 9** に示す．投入されたタスクが既存のスライスに割り当てられない場合，新しいスライスが作成される (図 9 下段の Slice 4)．このとき新しいスライスは投入されたタスクのみではなく，他のスライスにあるタスクで新しいスライスにも割当て可能なタスクをできるだけ multiple として割り当てる (図 9 下段の A", C", D")．ここではすでに multiple であるタスクより single であるタスクを優先とし，できるだけ multiple タスクを増やす．このため図 9 上段で B" を multiple せずに D を multiple にしている．single タスクをできるだけ減らすことで，スライスを削除できる可能性を大きくする．

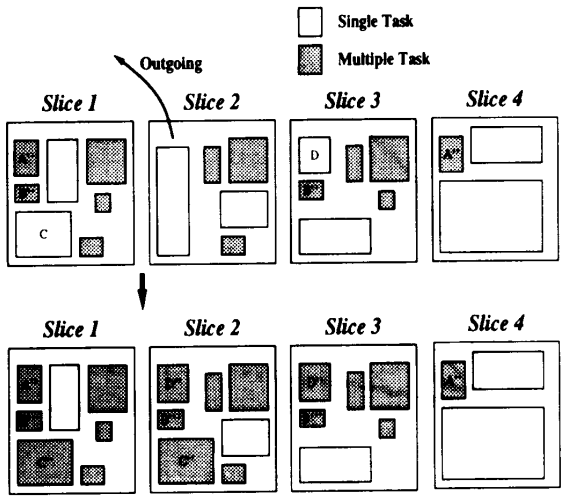

図10 タスクの終了
Fig. 10 Task finish.

タスクが終了したとき，そのタスクが占有していた領域に他のスライスのタスクが multiple として移る動作を図10に示す（下段の B"', C", D"）．この際に single であったタスクを優先して multiple にすることで，ラウンドロビンが一周するうちに処理される機会を多く与える（図10上段で A" より D が優先される）．また，ラウンドロビンを早くするため，single タスクがなくなったスライスは削除されるが，その機会をできるだけ増やす効果もある．

図11ではタスクが終了したとき，そのスライスに他のスライスからタスクが multiple として移り，あるスライスの single タスクがなくなった例を示す．このとき single タスクがなくなったスライス（スライス1）は削除され，そのスライスに multiple として存在していたタスクは縮退し，他のスライスに移る（図11中段の A"', B", C"', D", E", F"'）．この結果 C", E", F" は single タスクになる．

5. 性能評価

本論文で提案する TSS の性能評価のために，シミュレーションを行った．個々のタスクの到着間隔（$T_{interval}$），処理時間（$T_{service}$）は M/M/1 型待ち行列に従うと仮定する．ここで用いる M/M/1 型待ち行列とは，到着間隔，処理時間がそれぞれ指数分布に従い，タスクを受け付ける窓口が1つのモデルである．

本論文では，TSS化した2DB，AS（以後，TSS/2DB，TSS/AS と表記）と通常の2DB，AS でタスクを割り当てた場合について比較した．対象となる並列計算機は，32 × 32 台のメッシュ結合とした．シミュレーションでは仮想的時間を用い，タスクの平均投入間隔（$\overline{T}_{interval}$）を10から100まで変化させた際の状態について調べた．この際，平均処理時間（$\overline{T}_{service}$）が一定(100)であるため，計算機の負荷

$$Load = \overline{T}_{service}/\overline{T}_{interval} \quad (1)$$

は1から10となる．負荷1は1つのタスクの処理中に1つのタスクが投入される状態，負荷10は1つのタスクの処理中に10のタスクが投入される状態である．

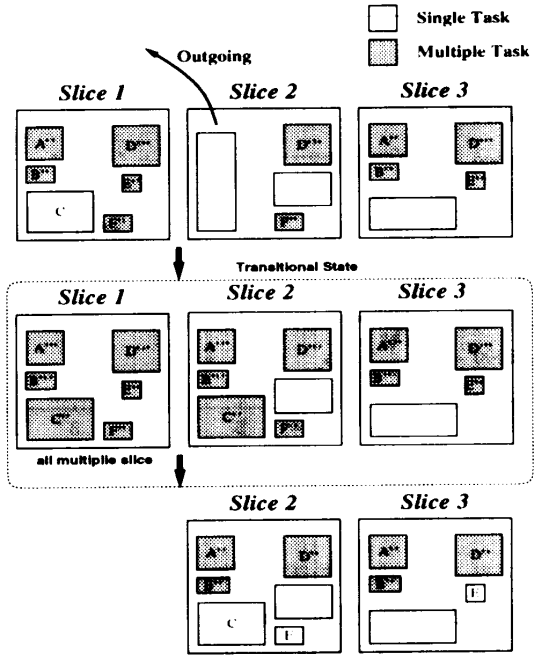

図11 スライスの減少
Fig. 11 Slice elimination.

タスクの形状は X 方向，Y 方向とも1から32までの一様乱数に従う長方形とした．ここでタスクが必要とするプロセッサ数は2の冪乗が最も使われると予想されるが，10や12がよく使われたり，3×3，5×5 などの正方数を要求するタスクも少なくない[9]．また多くのシミュレーション[4),6),7),10)]で形状の一辺を一様乱数に従う分布で行っているため，これらと比較するために本実験でも形状の一辺が一様乱数従うものとする．

以上の性質のタスクを100個投入し，全タスクの終了までシミュレートし，その性能を調べた．

5.1 全タスクの終了時間

図12に各負荷ごとにかかった全タスクの処理時間を示す．この図より，いずれのパーティションアルゴリズムでも TSS 化した場合の方が処理時間が短くて済むことが分かった．これは TSS 化した方がタスクの処理能力が高まることを示している．個々の処理能力を比較すると TSS/AS，AS，TSS/2DB，2DB の順となり，負荷2.0以上の場合 TSS 化の効果は TSS/AS で約15%，TSS/2DB で約12%であった．

また，ある程度高い負荷状態では全タスクの終了時間が一定時間に落ち着くが，TSS 化した方が高い負

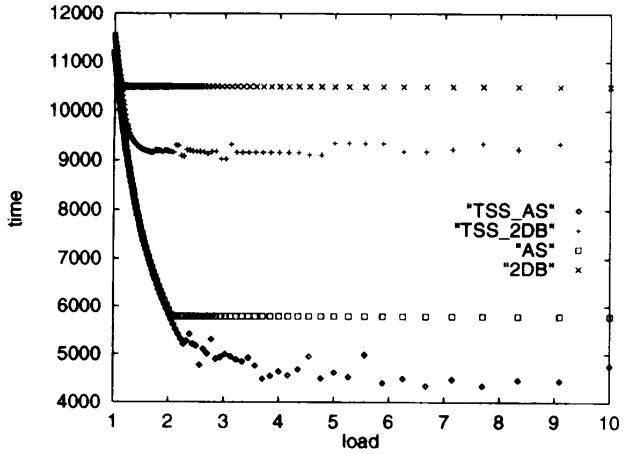

図12 全タスクの処理にかかる時間
Fig. 12 Completion time for all tasks.

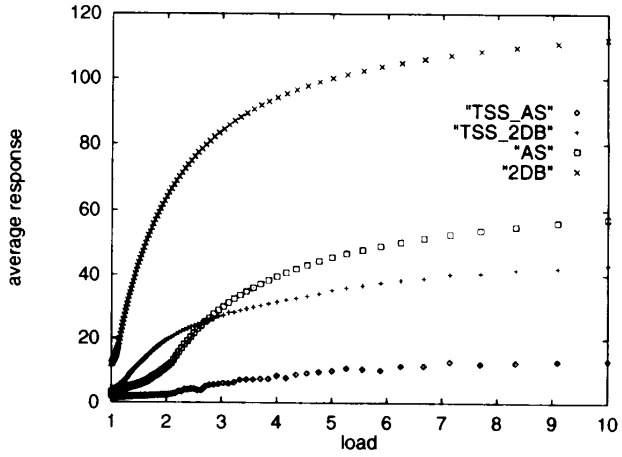

図13 平均応答
Fig. 13 Average response.

荷までタスクの処理時間の短縮が見られる．2DB では負荷 1.2，TSS/2DB では 1.6 まで，AS では 2.0，TSS/AS では 3.5 まで処理時間の短縮が見られる．これは TSS 化した場合の方が高負荷に強いことを示している．

TSS 化した場合に高い負荷で処理時間が落ち着くが，その際通常のパーティショニングアルゴリズムのように処理時間が固定せず，若干の変動が見られる．これは，パーティショニングアルゴリズムでは FCFS で処理しているため，ある程度負荷が高いと投入されたタスクが待ち行列で割当て待ちとなり，負荷の影響を受けなくなるためである．TSS では処理時間に変動が起こる理由は，投入時刻の違いによりスライス上へのタスク配置の組合せが異なり，この組合せで処理能力が異なるためである．

5.2 応　　答

図 13 に各負荷状態におけるタスクの平均応答を示す．応答（Repsonse）は，投入したタクスの到着時間（$T_{arrival}$），終了時間（T_{finish}）とした際に

$$Response = (T_{finish} - T_{arrival})/T_{service} \quad (2)$$

で示される．これはタスクが必要とする処理時間から何倍の時間をかければタスクの処理が終了するかを示すものであり，この値が高いほど待ち時間が長いことを示す．平均応答は個々のタスクの応答の平均をとったものである．

図 13 より，いずれのパーティショニングアルゴリズムも TSS 化した方が応答が良いことが分かる．この性質は TSS の特徴として当然のことであるが，TSS/2DB と AS は負荷 2.8 を境に AS の応答が TSS/2DB の応答より良くなっている．この性質は，5.1 節で明らかなように AS の方が TSS/2DB より処理能力が勝っているためである．負荷が高い状態では，TSS の特徴よりパーティショニングアルゴリズムの処理能力が支配的になることを示している．

個々のタスクの振舞いを調べるため，**図 14** に投入したタクスの到着時間（$T_{arrival}$），処理開始時間（T_{start}），終了時間（T_{finish}）を示したグラフを示す．点線はタスク到着時間から処理開始までを示し，太線は処理開始から終了までを示す．X 軸は経過時間，Y 軸は投入時間を示す．グラフは負荷 4.0 の場合である．この図より，TSS はタスクの処理が投入と同時に開始されることが分かる．また通常のパーティショニングアルゴリズムでは，投入時間が遅いほど処理の開始時間が遅いことが分かる．これは FCFS により，タスクの到着から処理開始までがタスクの大きさに関係なく，計算機の負荷状態によって割当て待ちが起こるためである．このため処理時間をあまり必要としないタスクも長い時間割当てを待たなければならず，応答が悪くなる．

図 15 はタスクが必要とする処理時間と応答の関係を示す．この図より通常のパーティショニングアルゴリズムでは，処理時間が短いタスクほど応答が高くなっていることが分かる．図 15 ではそれぞれの方式による応答を比較するため，応答の値を 200 まで表示したが，2DB と AS では処理時間が短いタスクでは 200 を超える応答を示す場合があった．2DB での最大応答は 813.5，AS での最大応答は 329.7 であった．TSS では処理時間が短いタスクが若干応答が大きくなるが，あまり影響を受けないことが分かる．これは図 14 からも分かるように，TSS と FCFS の性質の違いである．

5.3 プロセッサ利用率

図 16 に TSS/2DB と TSS/AS の各負荷状態におけるプロセッサ利用率を示す．ここでは X 軸が利用率，Y 軸が負荷を示す．プロセッサ利用率は全タスク

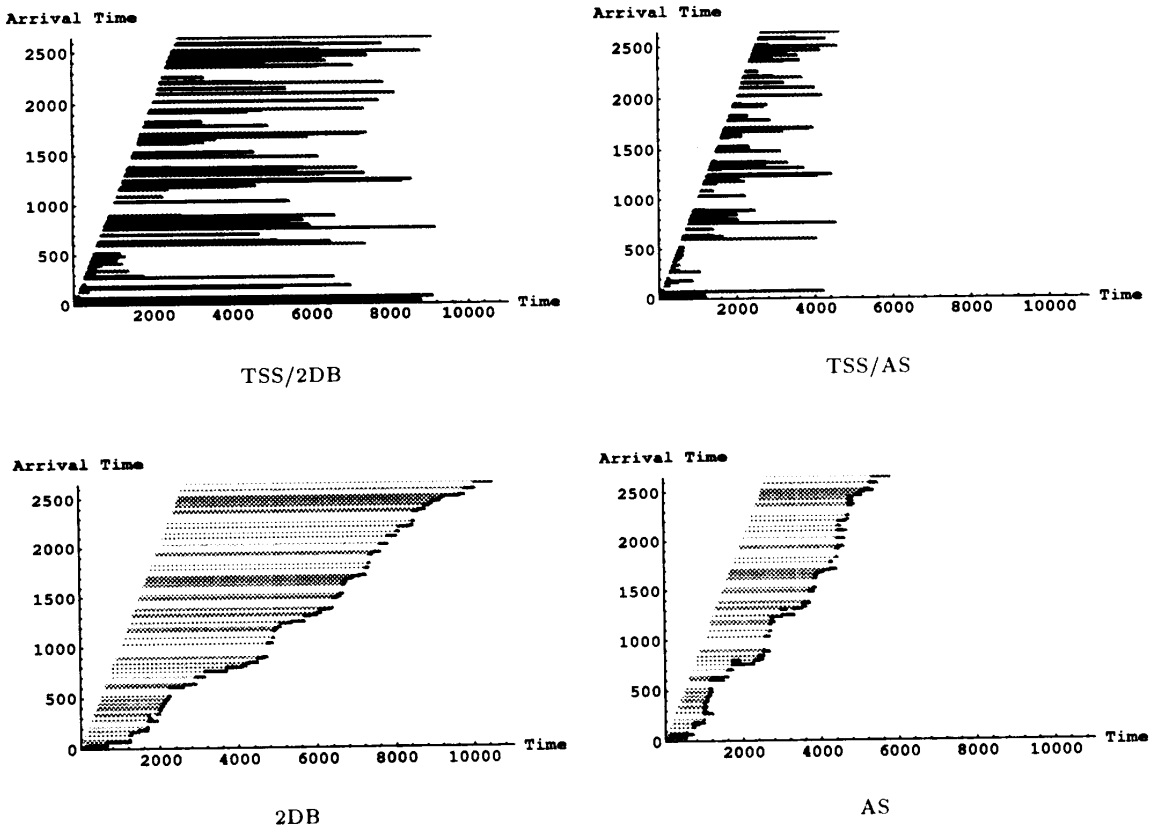

図 14 タスクが処理される過程：点線はタスク到着時間から処理開始まで，太線は処理開始から終了までを示す．

Fig. 14 Trace of each task processing : the dotted lines indicate the time between task arrival and task invoking and the wide lines indicates the time between task invoking and task finished.

終了までに稼働したプロセッサの割合である．ここでは投入するタスクは変えないため，プロセッサ利用率の値は 5.1 節で示した全タスクの終了時間の逆数に比例する．図 16 では，single タスクとして稼働したプロセッサと multiple タスクとして稼働したプロセッサに分けて表示した．このグラフより，TSS/2DB では multiple タスクの効果がほとんどなかったことが分かる．また，TSS/AS では負荷が 2.0 を超えてから multiple タスクの効果が顕著になることが分かる．その効果は TSS/2DB で約 1%，TSS/AS で約 6%であった．5.1 節の TSS 化による効果（TSS/2DB で約 12%，TSS/AS で約 15%）は，これらの multiple の効果も含んでいるが，閉塞状態を回避する効果の方がより有効であったことが分かる．

6. 関連研究

RWC の堀らは，二分木を元にしたタスクの管理による時分割処理方式を提案している[11),12)]．この方式は，計算機構成とは独立したタスク管理方法を利点としており，移植性に優れる．しかし，要求するプロセッサ数により 2 の巾乗数ごとの管理を行うため，2 の巾乗数に従わないプロセッサ数を要求するタスクはフラグメンテーションを生ずる．この性質はキューブ結合した計算機には適するが，メッシュ，トーラスなどの結合には適さない．

Pennsylvania 州立大の Yoo らは，本論文で提案した時分割処理化と似た方式を提案している[10)]．彼らの方式も仮想並列計計算機を用意し，その仮想計算機にパーティショニングアルゴリズムによりタスク配置を行う．しかし，この方式では小さいタスクサイズの仮想計算機，細長いタスクサイズの仮想計算機，大きいタスクサイズの仮想計算機のようにタスクサイズによって割当てを行う仮想計算機を制限しているために効率的な割当てができない場合がある．また，本方式で提案した複数の仮想計算機に存在するタスクを許していない．このため，プロセッサ利用率は本方式の方が高いと思われる．

7. 考　　察

本シミュレーションではタスク切替えによるオーバ

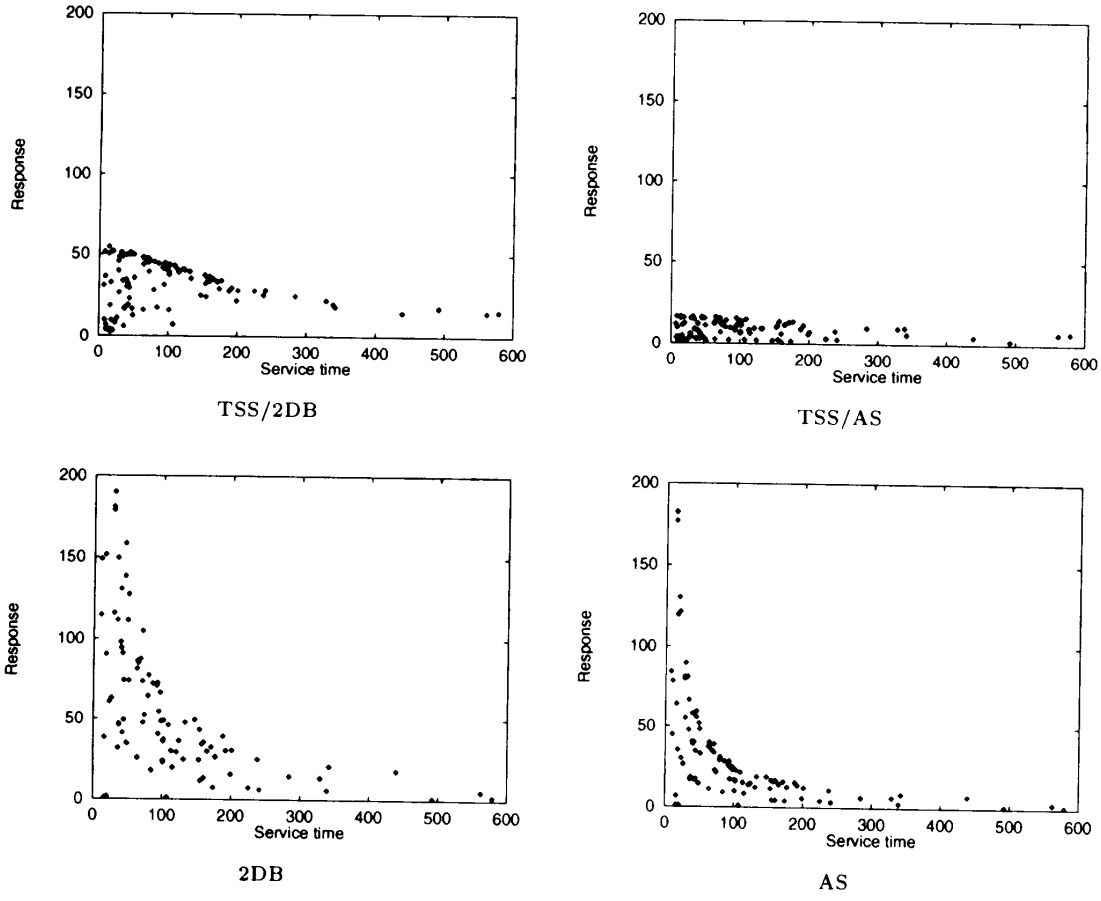

図 15 個々のタスクの応答
Fig. 15 Response for each task.

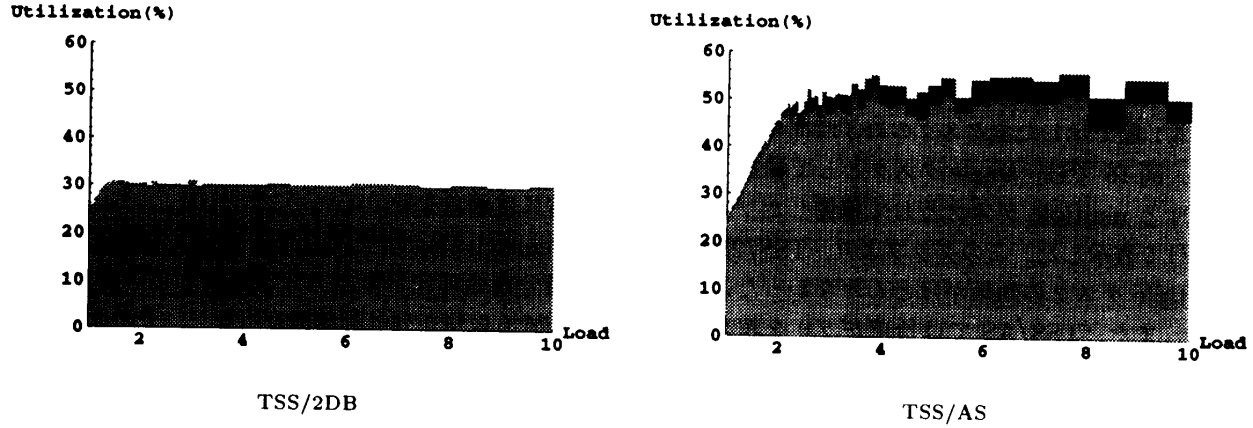

図 16 プロセッサ利用率：薄い灰色は single タスクしての稼働，濃い灰色は multiple タスクとしての稼働を示す．
Fig. 16 Processor utilization: light gray indicates single tasks and dark gray indicates multiple tasks.

ヘッドを考慮していないが，論文 13) で，オーバヘッドによる影響は時分割処理化によるプロセッサ利用率の向上で隠されることが示されている．本シミュレーションでも時分割処理化によるプロセッサ利用率の向上が示されており，同様の想定に従えば，TSS/2DB で 12%，TSS/AS で 15%までのオーバヘッドならば許容できることが分かる．

また，本シミュレーションでは時分割処理により，複数のタスクが 1 つのプロセッサに割り当てられてもメモリ不足が生じないと仮定した．この仮定は多くの論文で受け入れられており，実際に並列計算では多量のメモリが装備されているが，アプリケーションに

よっては適用できない場合がある．この問題を解決する手段として並列計算機上に仮想記憶を装備する方法が考えられる．筆者らも並列計算機でメモリの負荷分散を行う仮想記憶の研究を行っており[14]，この仮想記憶方式と組み合わせてアプリケーションに則した評価を行う予定である．

本論文では，パーティショニングアルゴリズムとして2DBとASを採用したが，他のパーティショニングアルゴリズムも容易にTSSに拡張できる．また，提案したTSSはメッシュ結合並列計算機を仮定しているが，他の結合形態の並列計算機にも容易に拡張可能である．他の結合形態に適用する場合，その結合形態に合ったパーティショニングアルゴリズムと重なり判定アルゴリズムを必要とする．これらは，すでにいくつか提案されているので，他の結合形態にもTSSを拡張していく予定である．

8. おわりに

本論文では，メッシュ結合計算機のパーティショニングアルゴリズムを拡張した時分割処理方式を提案した．提案した時分割処理方式では仮想並列計算機を用意し，これらのアルゴリズムを仮想並列計算機上でのサブメッシュの割当てに用いた．仮想並列計算機は一定時間ごとに実並列計算機で処理を進められる．本時分割処理方式では，ある仮想並列計算機でタスクが占有している領域を別の仮想並列計算機で未使用の場合，そのタスクが複数の仮想並列計算機に存在することでプロセッサ利用率を向上させた．

本論文では，一般性を示すため2つのパーティショニングアルゴリズム2D BuddyとAdaptive Scanを時分割処理化し，その効率向上をシミュレーションにより示した．この結果より，いずれのパーティショニングアルゴリズムの場合でもタスク全体の処理時間とプロセッサ利用率は時分割処理化により向上し，負荷が大きいときで10%以上向上することが分かった．タスクの応答も，パーティショニングアルゴリズムのみを使った場合より非常に良くなった．また，パーティショニングアルゴリズムのみでは，タスクの必要とする処理時間が小さいほど応答が悪くなる性質があったが，時分割処理方式での応答はタスクの必要とする処理時間に依存せず，ほぼ均一であった．これらの特徴を有する機能は多くのユーザに並列計算機の利用を許す場合，効率的にしかも均等に計算時間を割り当てることができるため，必要となるもののひとつである．

謝辞 本研究の一部はRWC計画の一環として行われたものである．関係各位に感謝いたします．

参考文献

1) 須崎，田沼，平野，一杉，塚本：メッシュ結合並列計算機のパーティショニングアルゴリズム2D-BuddyまたはAdaptive Scanを使った時分割処理，情報処理学会研究会報告，94-OS-65, pp.137-144 (1994).
2) Thinking Machines Corporation: Connection Machine CM-5 Technical Summary, Thinking Machines (1992).
3) Li, K. and Cheng, K.: Job Scheduling in a Partitionable Mesh Using a Two-Dimensional Buddy System Partitioning Scheme, *IEEE Trans. on Parallel and Distributed Systems*, Vol.2, No.4, pp.413-422 (1991).
4) Li, K. and Cheng, K.: A Two Dimensional Buddy System for Dynamic Resource Allocation in a Partitionable Mesh Connected System, *Journal of Parallel and Distributed Computing*, No.12, pp.79-83 (1991).
5) Ding, J. and Bhuyan, L.: An Adaptive Submesh Allocation Strategy for Two-Dimensional Mesh Connected Systems, *Proc. International Conference on Parallel Processing*, pp.193-200 (1993).
6) Chuang, P. and Tzeng, N.: An Efficient Submesh Allocation Strategy for Mesh Computer Systems, *Proc. 11th International Conference on Distributed Computing Systems*, pp.259-263 (1991).
7) Zhu, Y.: Efficient Processor Allocation Strategies for Mesh-Connected Parallel Computers, *Journal of Parallel and Distributed Computing*, Vol.16, pp.328-337 (1992).
8) Sedgewick, R.: *Algorithms*, Addison Wesley (1988).
9) Lo, V., Miller, J., Windisch, K., Zhuge, Y., Feitelson, D., Moore, R. and Nitzberg, B.: A Comparison of Workload Traces from Two Production Parallel Machines, *Supercomputing'95* (1995).
10) Yoo, B., Das, C. and Yu, C.: Processor Management Techniques for Mesh-Connected Multiprocessors, *Proc. International Conference on Parallel Processing*, pp.II-105-112 (1995).
11) 堀，石川，小中，前田，友清：超並列オペレーティングシステムにおけるスケジューリングの提案，情報処理学会研究会報告，94-OS-63, pp.25-32 (1994).
12) Hori, A., Ishikawa, Y., Konaka, H., Maeda, M. and Tomokiyo, T.: A Scalable Time-Sharing Scheduling for Partitionalble, Distributed Memory Parallel Machines, *Proc. 28th Annual Hawaii International Conference on*

System Sciences, pp.173-182 (1995).
13) Suzaki, K., Tanuma, H., Hirano, S. and Ichisugi, Y.: A Time Sharing System Scheme that Uses a Partitioning Algorithm for Mesh-Connected Parallel Computers, *7th IEEE Symposium on Parallel and Distributed Processing* (1995).
14) 平野，田沼，須崎：超並列システム用OS「超流動OS」における大域的仮想仮想記憶，*JSPP'93*, pp.237-244 (1993).

付録 パーティショニングアルゴリズム

投入されるタスクの大きさをX方向 w，Y方向 h としたときの2D BuddyとAdaptive Scanによるサブメッシュの割当てと解放の手続きを次に示す．

A.1 2D Buddy

タスクの割当て手続き

(1) Set $i \leftarrow \max(\lceil \log w \rceil, \lceil \log h \rceil)$.
(2) $j \geq i$ のうち最小の j となる FBL_j からbuddy要素を探す．なかったら，"not found" をリターン．
(3) While $j > i$, FBL_j からbuddy要素の1つ k をとり，それを崩して $(k \times 4, k \times 4 + 1, k \times 4 + 2, k \times 4 + 3)$ を FBL_{j-1} に加える．Set $j \leftarrow j - 1$.
(4) i レベルのbuddy要素を1つとりだし，それを返す．

タスクの解放手続き

(1) 解放するBuddy要素 k を該当するサイズの FBL_i に戻す．
(2) k が属するBuddy要素4つが FBL_i になかったらリターン，あったら以下の手続きでより大きいサイズとして戻す．
  (a) FBL_i から4つのbuddy要素を削除．
  (b) $k \leftarrow$ (削除したうち最小のbuddy要素 /4).
  (c) $i \leftarrow i + 1$ として，ステップ1へ．

A.2 Adaptive Scan

タスクの割当て手続き

flag = FALSE として呼び出す．

(1) If *flag* = FALSE then $T \leftarrow T(w, h)$, $a \leftarrow \max(0, M - w + 1)$, $b \leftarrow \max(0, N - h + 1)$. otherwise $T \leftarrow T(h, w)$, $a \leftarrow \max(0, N - h + 1)$, $b \leftarrow \max(0, M - w + 1)$.
(2) T と割り当てられているサブメッシュに基づいてcoverage setを作る．$x \leftarrow 0$, $y \leftarrow 0$.
(3) (x, y) がcoverage setに含まれなければ，ステップ(4)へ．
  (a) If $x < a - 1$ then $x \leftarrow d + 1$ (d は (x, y) を含むcoverage setの x 方向の最大値)，ステップ3へ．
  (b) If $x = a - 1$ and $y < b - 1$ then $x \leftarrow 0$, $y \leftarrow y + 1$, ステップ3へ．
  (c) If $x = a - 1$ and $y = b - 1$ and *flag* = FALSE, then *flag* $\leftarrow$ TRUE, ステップ1へ．そうでなかったら "not found" を返す．
(4) if *flag* = FLASE then $S \leftarrow (x, y, w - 1, h - 1)$; otherwise $S \leftarrow (x, y, h - 1, w - 1)$, リターン．

タスクの解放手続き

(1) サブメッシュを解放．リターン．

(平成7年9月 4 日受付)
(平成8年4月12日採録)

須崎 有康（正会員）

1965年生．1990年東京農工大学大学院数理情報工学専攻修士課程修了．1991年同大学大学院工学研究科電子情報工学専攻博士後期課程中退．同年電子技術総合研究所入所．並列計算機用オペレーティングシステム，プログラムのパフォーマンスチューニング，アルゴリズムアニメーションに興味を持つ．

田沼 均（正会員）

1961年生．1986年東北大学大学院工学研究科電気及び通信工学専攻修士課程修了．同年電子技術総合研究所入所．現在，超並列計算機のオペレーティングシスムの研究に従事．

平野 聡（正会員）

1962年生．1985年電気通信大学材料科学科卒業．1987年同大学大学院経営工学専攻修了．1992年東京大学大学院博士課程情報工学専攻修了．博士（工学）．同年電子技術総合研究所入所．並列分散処理とネットワーク情報社会のアーキテクチャに興味を持つ．IEEE会員．

一杉 裕志（正会員）

1965年生．1990年東京工業大学大学院情報科学専攻修士課程修了．1993年東京大学大学院情報科学専攻博士課程修了．博士（理学）．同年電子技術総合研究所入所．オブジェクト指向言語，並列・分散言語，リフレクションに興味を持つ．日本ソフトウエア科学会，ACM各会員．

塚本 享治（正会員）

1949年生．1972年東京大学計数工学科卒業．同年電子技術総合研究所入所，現在同分散システム研究室長．知能ロボット用計算機システムの研究の後，オブジェクト指向型分散システム，並列処理システムなどの研究開発に従事．1989年日本ロボット学会論文賞，1994年科学技術長官賞研究功績賞．現本学会理事．