システムソフトウェアと
オペレーティング・システム 73－16
（1996．8．27）

連続型と非連続型を融合したタクス配置アルゴリズムの提案

須崎 有康　田沼 均　平野 聡　一杉裕志　Chris Connelly　塚本亨治

suzaki@etl.go.jp

電子技術総合研究所

連続型タクス配置アルゴリズムと非連続型タクス配置アルゴリズムを融合したアルゴリズムを提案する。提案するアルゴリズムは、連続型で問題になる低いプロセッサ利用率を改善し、非連続型で問題になる長い通信経路とメッセージ衝突を緩和する。本論文ではメッシュ結合並列計算機を対象として、連続型タクス配置アルゴリズム Adaptive Scan と非連続型タクス配置アルゴリズム Multi Buddy を融合したアルゴリズムを示す。更にその性能向上を確かめるため、メッセージ転送方式として一般に普及してる Wormhole、メッセージの転送パターンとして最もメッセージ衝突を起こしやすい All to All を対象にシミュレーションを行なった。この結果、提案したアルゴリズムが既存のものより計算機全体の効率的利用と個々のユーザへ提供できるサービスの両面で性能向上があったことを示す。

A Task Allocation Algorithm which combines a contiguous and non-contiguous one

Kuniyasu SUZAKI, Hitoshi TANUMA, Satoshi HIRANO,
Yuuji ICHISUGI, Chris Connelly, and Michiharu TSUKAMOTO

Electrotechnical Laboratory
1-1-4 Umezono,Tsukuba-city,Ibaraki,305,Japan

We propose a task allocation algorithm that combines a contiguous task allocation algorithm and a non-contiguous task allocation algorithm. The proposed algorithm settles the problems which caused by contiguous task allocation algorithms and non-contiguous task allocation algorithms; low processor utilization and long distance of message transfer. In this paper we target for mesh-connected parallel computers and show a task allocation algorithm that combine Adaptive Scan and Multi Buddy. To show the performance improved by the proposed algorithm, we conducted simulation runs under an assumption of wormhole routing and all-to-all message pattern. From the results we knew that the proposed algorithm improved utilization of parallel computer and service for each task.

1 はじめに

現在、並列計算機の効率的稼働と個々のユーザに対して応答性の良いサービスを目指してタスク配置アルゴリズムの研究が広く行なわれている。タスク配置アルゴリズムはタスクの並列アルゴリズムが要求するプロセッサ数に従って、FCFS(First Come First Serve)で物理プロセッサに効率良く割り当て、空間共有(Space Sharing)型のマルチタスクを実現する。このアルゴリズムによってプロセッサをあまり使わないタスクが並列計算機全体を独占することがなくなり、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。また、個々のタスクも他のタスクと同時実行可能になるため、応答性も向上する。

タスク配置アルゴリズムは、タスクの並列アルゴリズムが要求するプロセッサ形状を保持するか、しないかにより、連続型と非連続型に分けられる。タスク配置アルゴリズムが提案された初期には、対象並列計算機に適合するプロセッサ形状をタスクが要求し、その形状が保持されることが絶対条件とされた。この性質は閉パーティショニングと呼ばれる。例えばメッシュ結合並列計算機ではメッシュのプロセッサ形状を要求し、キューブ結合並列計算機ではキューブのプロセッサ形状を要求した。閉パーティショニングによりタスク内のメッセージ交換は領域内のネットワークのみを使い、他のタスクと干渉することがない。つまり連続型タスク配置アルゴリズムはタスクの割り当て以後、タスクに何の影響も及ぼさないことが保証できた。

しかしオレゴン大のLiuらは論文[5]でタスク間のメッセージ干渉の頻度が予想されていた量よりは多くなく、またその遅延はWormholeやVitrual Cut Through等のメッセージ転送方式によって小さくなることを示した。このため、メッセージ干渉を完全に削除するよりも、プロセッサ形状を壊してプロセッサ利用率を向上させた方が全体のプロセッサ利用率も個々のタスクの応答性も向上できるとし、プロセッサ形状を必ずしも保たずに割り当てる非連続型タスク配置アルゴリズムを提案した。

この非連続型タスク配置アルゴリズムは通信量が少ない場合に連続型タスク配置アルゴリズムより良い性能を得たが、通信量が多くメッセージ衝突が頻繁になる場合は連続型タスク配置アルゴリズムより性能が悪くなった。筆者らは、この欠点がLiuらの提案した非連続型タスク配置アルゴリズムでは閉パーティショニングを保って割り当て可能な領域が存在しても、プロセッサ形状を崩してしまうことから来ると考えた。そこで、プロセッサ形状が保存できる際には形状を保って割り当て、形状が保存できなくなってはじめてプロセッサ形状を崩すアルゴリズムを提案する。本論文では、この有効性を確かめるため、メッセージ転送の遅延をシミュレートできるシミュレータprocsimity[5]を使い、その効果を確認した。

以下2章で今まで提案されている連続型と非連続型のタスク配置アルゴリズムについて概観する。3章では筆者らが提案する連続型と非連続型の融合方式を説明する。

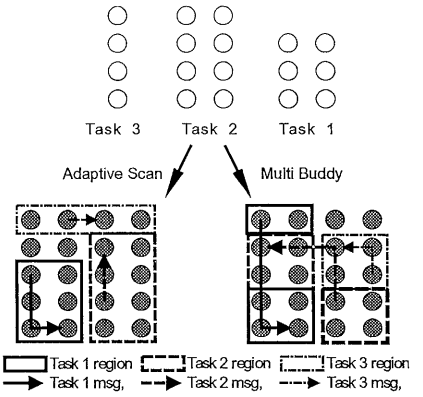

図 1: 連続型と非連続型のタスク配置

4章では提案するタスク配置アルゴリズムの実行効率をシミュレーションの結果で示す。最後に5章で結論を述べる。

2 タスク配置アルゴリズム

タスク配置アルゴリズムは対象とする計算機アーキテクチャ毎に提案されている。ここではメッシュ結合並列計算機を対象にその連続型と非連続型のタスク配置アルゴリズムを概観する。ここでプロセッサの位置はX方向、Y方向により(x,y)で示され、左下を(0,0)とする。

2.1 連続型

連続型タスク配置アルゴリズムは、タスクが要求するプロセッサ形状を崩さずに割り当て可能領域を探索するアルゴリズムである。タスク内でのメッセージは割り当てられたプロセッサを繋ぐネットワークのみを使用し、他のタスクのメッセージと干渉することがない。この性質は閉パーティショニングと呼ばれ、連続型タスク配置アルゴリズムではこの性質が保証される。メッシュ結合並列計算機を対象とする連続型タスク配置アルゴリズムにはFrame Slide[1]、First Fit[8]、Best Fit[8]、Adaptive Scan[2]、2D Buddy[3, 4],Quick Allocation[7], Busy List[6]などがある。本論文では、このうちプロセッサ利用率が高く実装も簡単なAdaptive Scan(AS)を取り上げる。

AS[2]ではタスクが要求する長方形のプロセッサ形状をメッシュの下端(0,0)からX方向にサーチし、他のタスク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方向に一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領域が見つからないとき、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。図1に記述されたASの割り当て例では、TASK3がその方向を変えている。ASは比較的高いプロセッサ利用率を示すが、例えばTASK3が2×3のプロセッサ形状を要求する場合ではその形状の空き領域がないため、たとえ未使用プロセッサが6個あっ

ても割り当てを行なえない。

2.2 非連続型

非連続型タスク配置アルゴリズムは、タスクが要求するプロセッサ形状は考慮せずに、要求されるプロセッサ数が未割当として残っていればタスクを割り当てるアルゴリズムである。ここでは閉パーティショニングが保証されず、タスク間のメッセージの衝突を許す開パーティショニングとなる。非連続型タスク配置アルゴリズムはまだその数が少なく、論文[5]でRandom, Page, Multi Buddyが提案されている。これらはプロセッサ利用率の点では全て同じ性能を示すが、メッセージ距離を考慮するとタスク形状をあまり崩さないMulti Buddyがもっとも短く、メッセージ衝突が少ない。本論文では、このMulti Buddy(MB)を対象にする。

MBは連続型タスク配置アルゴリズム2 Dimensional Buddy(2DB)[3, 4]を非連続型に拡張したものである。2DBでは並列計算機より小さい2の巾乗の正方形を基本単位として割り当てる。これはBuddy要素と呼ばれ、正方形の大きさごとに管理される。$2^n \times 2^n$の大きさのBuddy要素は、その左下のプロセッサ位置が$(2^n \times p, 2^n \times q)$から始まる。Buddy要素はその位置により番号が振られ、その番号によって管理される。この2DBでは、タスクが要求するプロセッサ形状が$W \times H$とすると一辺が$2^{\lceil max(log_2 W, log_2 H)\rceil}$のBuddy要素を与える。このため、WとHが共に$2^{\lceil max(log_2 W, log_2 H)\rceil}$でない際、与えられたBuddy要素には使用しないプロセッサ群も含まれ、フラグメンテーションを起こす。

MBでは2DBと同様に2の冪乗の正方形のBuddy要素を使うが、フラグメンテーションを起こさないようにタスクが要求するプロセッサ形状を変形する。ここでは要求されたプロセッサ数のみに注目し、$2^n \times 2^n$の正方形の領域に区切って、各正方形の大きさ毎に割り当てを行なう。例えばTASK1では2×3を要求しているが、これは$6 = (2^1 \times 2^1) \times 1 + (2^0 \times 2^0) \times 2$とし、$2^1 \times 2^1$のBuddy要素一つと$2^0 \times 2^0$のBuddy要素二つが割り当てられる。また、TASK1を最初に割り当てられる際にAdaptive Scanが割り当てた連続領域が空いているが、Multi Buddyではできるだけ大きい領域を崩さない戦略のため、近くの領域が使用可能でも通信距離の遠い断片領域を割り当てる。このため図1のTASK1の様に遠いプロセッサ同士がメッセージ交換を行なわなければならず、また他のタスクの通信経路も使用するためメッセージ衝突も起こす。

3 連続型と非連続型の融合

連続性タスク配置アルゴリズムと非連続型タスク配置アルゴリズムがそれぞれ持つ欠点を克服するために、我々は両者を併用したタスク配置アルゴリズムを提案する。本論文では、連続型タスク配置アルゴリズムの中でプロセッサ利用率の高いAdaptive Scan(AS)と非連続型タスク配置アルゴリズムの中で通信距離を比較的短くするMulti Buddy(MB)を融合した。

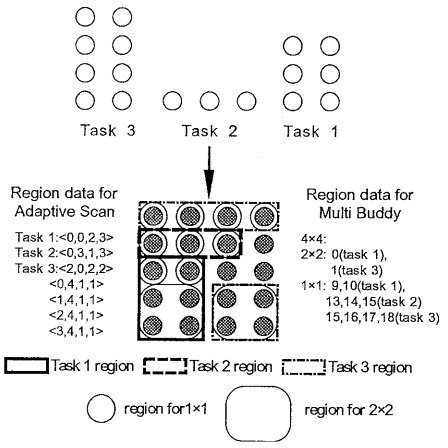

図2: 提案するアルゴリズムによる配置

3.1 提案するアルゴリズムの概要

提案するタスク配置アルゴリズム(AS&MB)は、タスクが投入された際にまずASで領域探索を行ない、これが失敗した場合にMBに切替える。このためプロセッサ形状保存ができる場合はそのまま領域を割り当て、形状が保存できなくなってはじめてタスクのプロセッサ形状の分割を行なう。これによりプロセッサ利用率を損なわず、割り当て領域をできるだけ近接させることができる。

AS&MBによる割り当て例を図2に示す。この例では、TASK1とTASK2はASによって割り当てられ、TASK3はMBによって割り当てられている。TASK3は$2 \times 4 = (2^1 \times 2^1) \times 2$と認識され、$2^1 \times 2^1$のBuddy要素を二つ要求する。しかし、TASK1とTASK2を割り当てた後の領域には、$2^1 \times 2^1$のBuddy要素が一つのみのため、もう一つの$2^1 \times 2^1$の領域は$2^0 \times 2^0$のBuddy要素四つとして割り当てられる。

配置してあるタスクの領域管理は、ASでは個々のタスクの配置位置を示す四つ組のデータ構造$< x, y, w, h >$で表現される。これは、タスクが割り当てられた領域の左下の(x,y)座標とタスクの縦横寸法の$w \times h$である。ASで割り当てたTASK 1とTASK 2はそれぞれ$< 0,0,2,3 >$と$< 0,3,1,3 >$となる。TASK 3はMBで不連続領域が割り当てられているため、その構成は割り当てられたBuddy要素を表す複数の四つ組データ$< 2,0,2,2 >$,$< 0,4,1,1 >$,$< 1,4,1,1 >$,$< 2,4,1,1 >$,$< 3,4,1,1 >$となる。

MBによる使用領域の管理は、全てBuddy要素のサイズ毎に行なわれていて、各要素にはその位置により、Buddy番号が振られている。TASK1とTASK2はASによって割り当てられているため、その領域をBuddy表現に直さなければならない。

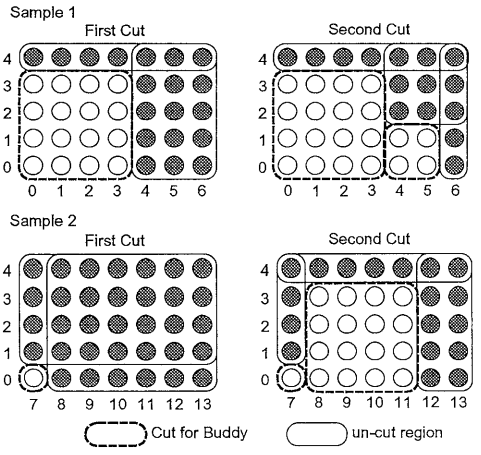

図 3: Buddy 表現への変換

3.2 タスク管理データの相互変換

上記で説明したように、AS と MB では並列計算機に割り当てているタスク領域を管理するデータ構造が異なる。AS&MB では AS と MB の両方で探索を行なうため割り当てた領域情報を相互に変換する必要がある。

MB で割り当てられた領域は上記で示した通り、簡単に AS の領域管理データに変換できる。但し、一つのタスクに対し複数の連続領域を持つように変更しなければならない。

AS で割り当てられた領域 $< x, y, w, h >$ は、大きさだけではなく位置によっても切り出される Buddy 要素が異なる。$< x, y, w, h >$ を Buddy 候補領域とするとその左下の位置から切り出せる Buddy 要素の大きさは、そのプロセッサ位置とタスクサイズの大きさから求まる。そのアルゴリズムは、割り当て領域の左下のプロセッサ位置 (x,y) を

$x = 2^n \times l_n + 2^{n-1} \times l_{n-1} + \cdots + 2^{n-m} \times l_{n-m}$

$y = 2^q \times p_q + 2^{q-1} \times p_{q-1} + \cdots + 2^{q-r} \times p_{q-r}$

で表現し、もっとも小さい 2 の冪乗の構成要素 $2^{n-m}, 2^{q-r}$ のうち、小さいものが (x,y) を左下とする最大の Buddy 要素候補となる（例外として、値が 0 の時には任意の 2 の冪乗数が候補となる）。更に両辺を構成する最大の 2 の冪乗数 $2^{(\lceil log_2 w \rceil)}$ と $2^{(\lceil log_2 h \rceil)}$ と供に比較し、三者のうちもっとも小さいものが Buddy 要素とする。

Buddy 要素を切り出した後に残った領域は再び Buddy 候補領域とし、サブセットを持たない長方形で構成される。これらの Buddy 候補領域はどちらを始点とした方が大きな Buddy を切り出せるかわからないため、重なることがある。これらの Buddy 候補領域に対し同様の手順を繰り返し、Buddy 候補領域がなくなるまで行なう。

この切り出し例を図 3 を用いて説明する。図 3 の sample1 の First Cut では $< 0, 0, 7, 5 >$ から Buudy 要素を探す。左下のプロセッサ位置が (0,0) のため、Buddy 要素はタスクサイズのみに依存する。7×5 に含まれる最大の 2 の冪乗は数 2^2 であり、$2^2 \times 2^2$ の Buddy 要素が切り出される。$< 0, 0, 7, 5 >$ の代わりに残った領域 $< 4, 0, 3, 5 >$ と $< 0, 4, 7, 1 >$ が次の Buddy 候補領域なる。この際、$< 4, 4, 3, 1 >$ の領域が重なっている。Second Cut では $< 4, 0, 3, 5 >$ から同様にして $2^1 \times 2^1$ の Buddy 要素が切り出され、$< 4, 0, 3, 5 >$ の代わりに、$< 6, 0, 1, 5 >$ と $< 4, 2, 3, 3 >$ が Buddy 候補領域になる。

また、同一サイズのタスクでも配置位置によって Buddy 表現が異なる。この例を図 3 の Sample2 に示す。Sample2 の First Cut では $< 7, 0, 7, 5 >$ から Buddy 要素を切り出す。左下のプロセッサが (7,0) であるため、この位置の最大 Buddy 要素は x の値 $7(2^2 + 2^1 + 2^0)$ に依存し、7 を構成する最小の 2 の冪乗数 $2^0 = 1$ が候補になる。タスクサイズは 7×5 なので、これを構成する最大 2 の冪乗数は 2^2 になる。しかしここでは、プロセッサの位置 (7,0) により最大 Buddy 要素は $2^0 \times 2^0$ に押えられてしまう。このため、$2^0 \times 2^0$ の Buddy 要素を切り出した残りの領域、$< 8, 0, 6, 5 >$ と $< 7, 1, 7, 4 >$ が次の Buddy 候補領域になる。Second Cut では $< 8, 0, 6, 5 >$ を対象にして $2^0 \times 2^0$ の Buddy 要素が切り出される。この際、切り出した領域がもう一つの Buddy 候補領域 $< 0, 1, 7, 4 >$ と重なっているため、この領域も崩され $< 0, 1, 1, 4 >, < 0, 4, 7, 1 >, < 12, 1, 2, 5 >$ が Buddy 候補領域になる。

4 シミュレーションによる評価

本論文で提案するタスク配置アルゴリズムの性能評価のために、シミュレーションを行なった。性能評価はオレゴン大で開発された procsimity[5] を改良して行なった。procsimity ではメッセージの転送方式に Store& Forward, Virtual Cut Through, Wormhole を備え、メッセージ衝突によるタスク処理の遅延をシミュレートできる。メッセージパターンも All to All, All to One, FFT, NAS Multigrid 等揃えており、各タスクの性質を変えて性能評価できる。本論文では、32×32 のプロセッサを想定し、1000 個のタスクを投入間隔を指数分布に従う乱数により変えて、その変化を調べた。タスクの平均処理時間は 1000 unit time とし、平均投入間隔は 100 ～ 1000 unit time とした。この時、計算機の負荷 (平均処理時間／平均投入間隔) は 1 ～ 10 となる。タスクは長方形のプロセッサ群を要求し、その一辺の変化は 1 から 32 の一様乱数とした。本論文では、メッセージの転送方式にもっとも良く使われている Wormhole(WH) を採用した。WH では 1byte flit とし、バッファは 1byte とした。1 flit のディレイは 3 unit time とした。また本論文では、メッセージパターンを通信量の最も多い All to All に固定し、一回のメッセージが 8byte、メッセージの回数を平均 40 個の指数分布の乱数で割り当てた。All to All はタスク内の任意のプロセッサがタスク内の全てのプロセッサにメッセージを送る転送パターンである。このパターンは任意のプロセッサ同士がメッセージ転送を行なうため、タスク内でもメッセージ衝突が起こる。論文 [5] の結果からこのメッセージパターンは非連続型タス

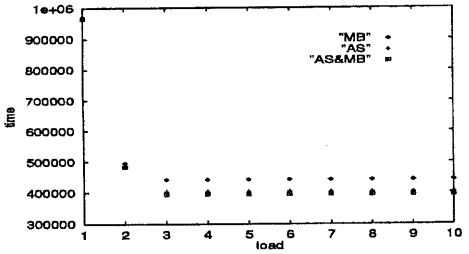

図 4: 全タスク終了時間

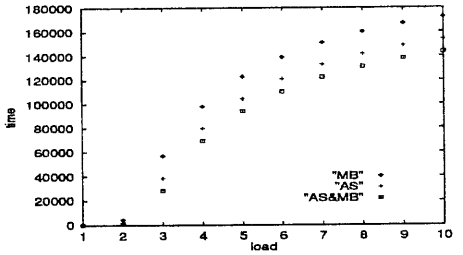

図 5: 割り当て待ちした平均時間

ク配置アルゴリズムが不得意とするものの一つであり、連続型が有利となることが知られている。

本論文では、システム全体が効率的に利用された程度を現す指標（全タスク終了時間）と個々のユーザへ提供できたサービスの程度を現す指標（割り当て待ち時間とレイテンシ）を評価する。さらに提案するタスク配置の振舞いを調べるため、通信距離とアルゴリズムの使用頻度の状況を示す。

4.1 全タスク終了時間

投入した千個のタスクが全て終了する時間を各負荷において調べた。図4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、負荷が低い状態ではどのタスク配置アルゴリズムを使っても差は見られないが、負荷が高くなると AS&MB,AS,MB の順に終了時間が早いことがわかった。このうち AS と MB の終了時間の差は All to All のメッセージパターンによるメッセージ衝突の頻度が大きいためであり、プロセッサの利用率よりメッセージ衝突による遅延の影響が大きいためであった。AS&MB では、AS よりメッセージ衝突が多く起こっているが、MB よりメッセージ衝突は少なく、高いプロセッサ利用率がそれを打ち消していた。メッセージ衝突の詳細は 4.2節で更に調べる。

このグラフより、更にメッセージの転送量を多くすれば通信距離の短い AS が AS&MB に優ると思われたが、実際にメッセージ量を多くしても AS&MB の優位は変らなかった。この理由は、メッセージ衝突による遅延を調べるとメッセージが平均 40 以上では既に飽和しており、AS によるメッセージ衝突の減少の効果はなかった。むしろ AS&MB の高いプロセッサ利用率の効果が現れ、AS よりも明らかに早い終了時間を示した。

4.2 個々のタスクへのサービス

個々のタスクへ提供できるサービスの程度を調べるため、その二つの要因となる割り当て待ちをした平均時間とメッセージのレイテンシの平均時間を測定した。

あるプロセッサ群にタスクが一度割り当てられたら、タスクが終了するまでそのプロセッサ群は他のタスクを割り当てられることがない。このため、ユーザへのサービスを妨げる要因の一つは、タスクが割り当てられなかった際にプロセッサの解放を待つ時間である。もう一つの要因は、割り当てられた後のメッセージのレイテンシ時間である。この二つの要因を調べることにより、タスク配置アルゴリズムの能力を比較した。

図5にタスクが割り当て待ちをした平均時間を示す。この図より、いずれのタスク配置アルゴリズムも負荷が2まではほとんど割り当て待ちの時間がなかったが、それ以上の負荷では負荷が大きくなるほど割り当て待ち時間が長くなった。待ち時間は AS&MB,AS,MB の順に短かった。筆者らはプロセッサ利用率の高い MB が AS より待ち時間が短いものと予想したが、All to All のメッセージパターンでは一つのタスクの処理時間がメッセージ衝突により長くなるため、メッセージ衝突の多い MB では個々のタスクの処理時間が AS より長くなった。そのため、プロセッサの空きを待ち時間が MB では長くなり、高いプロセッサ利用率でもその影響は埋めることができなかった。

この状況を更に詳しく調べるため、図6に個々のメッセージのレイテンシの平均時間を示す。この図より、AS は負荷の変化に関わらずレイテンシが一定であったことがわかる。これは閉パーティションの効果であり、一度割り当てられれば他のタスクと干渉することがないため負荷とは独立となった。MB と AS&MB は、AS よりもレイテンシが長くなった。これは割り当てが閉パーティショニングを保たないため通信距離が長くなったことと、他のタスクと干渉するためである。MB は AS&MB よりレイテンシが長い一つの要因は通信距離が長いように割り当てられるためである。通信距離については 4.3節で詳細を調べる。また負荷の影響は負荷が3までであり、それ以降は MB、AS&MB とも一定となっている。これは負荷が3以上ではそれぞれのタスク配置アルゴリズムの能力を越え、タスクが割り当てられないためメッセージ衝突は負荷から独立となった。

4.3 通信距離

通信距離の伸長を現す基準として、要求する領域に対して割り当てられた領域が乖離する程度を現す離心率 [5] がある。離心率は割り当てられた領域を取り囲む最小領域（割り当てられた領域の X 方向の最大距離と Y 方向の最大距離の積）と要求した領域の商で求められる。図2の TASK1 と TASK2 では要求する領域と割り当てられた領域が同一のため離心率は 1.0 となり、TASK3 では必要

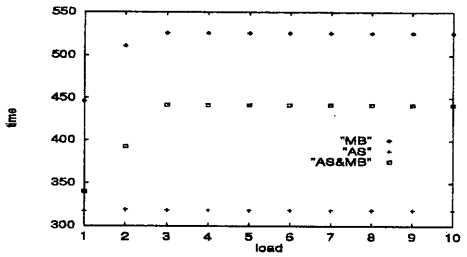

図 6: レイテンシの平均時間

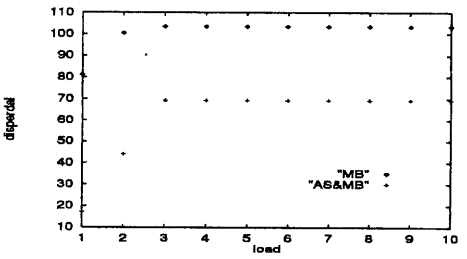

図 7: 離心率

とする領域は2×4であり、割り当てられた領域を取り囲む領域は4×5なので離心率は2.5となる。

MBとAS&MBの離心率の平均を図7に示す。この図よりAS&MBが各タスクを割り当てた領域はMB単体より近接したものになっていたことわかる。ここでも負荷が3以上では離心率が飽和しているが、これもそれぞれのタスク配置アルゴリズムの能力を越えたためである。

4.4 ASとMBの使用頻度

AS&MBの動的な振舞いを知るため、AS&MBで使用されるASとMBの使用頻度を求めた。図8はそれぞれの使用頻度を表している。

負荷が低い状態ではASによるタスク配置が支配的であるが、負荷が高い状態ではASとMBの使用度は半々になった。この原因は負荷が軽い時はプロセッサが余っているので分割する必要があまりなくASで十分割り当て領域を探すことができた。しかし、負荷が高くなると連続領域を探すことが難しくなり、MBが多く用いられるになった。負荷が高い飽和状態ではASとMBの利用は半々になった。

5 おわりに

本論文では連続型タスク配置アルゴリズムと非連続型タスク配置アルゴリズムを融合したタスク配置アルゴリズムを提案した。提案したタスク配置アルゴリズムは、連続型で問題となったプロセッサ利用率の低さと非連続型で問題となった通信距離の伸長を、両者を組み合わせることでお互いにその欠点を補うようにした。

更にその実際の有効性を確かめるためにメッシュ結合並列計算機の連続型タスク配置アルゴリズムAdaptive

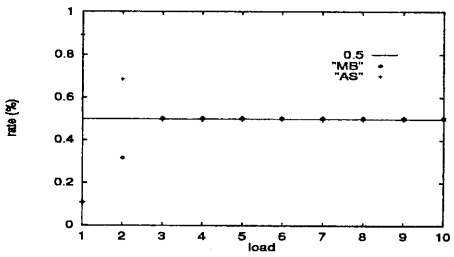

図 8: ASとMBの使用頻度

Scanと非連続型タスク配置アルゴリズムMulti Buddyを組み合わせたAS&MBを作成し、シミュレーションによりその効果を調べた。シミュレーション結果より、AS&MBはAS,MB単体よりも計算機全体の利用効率と個々のタスクに提供できるサービスの両面で単体の性能よりも良い結果を示した。

本手法はメッシュ結合並列計算機のみに適用できるばかりでなく、その他の結合形態の計算機のタスク配置アルゴリズムにも適応でき、同様の問題を解決できることが期待される。

謝辞

本研究の一部はRWC計画の一環として「超並列システムアーキテクチャに関する研究」で行なわれたものである。関係各位に感謝いたします。

参考文献

[1] P. Chuang and N. Tzeng. An Efficient Submesh Allocation Strategy for Mesh Computer Systems. *The 11th ICDCS*, 1991.

[2] J. Ding and L. N. Bhuyan. An Adaptive Submesh Allocation Strategy for Two-Dimensional Mesh Connected Systems. *ICPP*, 1993.

[3] K. Li and K. Cheng. 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,, 1991.

[4] K. Li and K. Cheng. A Two Dimensional Buddy System for Dynamic Resource Allocation in a Partitionable Mesh Connected System. *Journal of Parallel and Distributed Computing*, No. 12,, 1991.

[5] W. Liu, V. Lo, K. Windish, and B Nitzberg. Non-contiguous Processor Allocation Algorithms for Distributed Memory Multicomputers. *Supercomputing*, 1994.

[6] D. D. Sharma and D. K. Pradhan. A Fast and Efficient Strategy for Submesh Allocation in Mesh-Connected Parallel Computers. *The 5th SPDP*, 1993.

[7] S.M. Yoo and H.Y. Youn. An Efficient Task Allocation Scheme for Two-Dimensional Mesh-Connected Systems. *The 15th ICDCS*, 1995.

[8] Y. Zhu. Efficient Processor Allocation Strategies for Mesh-Connected Parallel Computers. *Journal of Parallel and Distributed Computing*, Vol. 16,, 1992.