システムソフトウェアと
オペレーティング・システム 63－4
（１９９４．３．２２）

超並列オペレーティングシステムにおける スケジューリング方式の提案

堀 敦史, 石川 裕, 小中 裕喜, 前田 宗則, 友清 孝志
技術研究組合 新情報処理開発機構 (RWC) つくば研究センタ

e-mail:{hori,ishikawa,konaka,m-maeda,tomokiyo}@trc.rwcp.or.jp

疎結合型の超並列マシンでは, 従来の OS が用いているような単一の待ち行列によるスケジューリングは性能上のボトルネックを生じる可能性がある. ここで提案する「分散樹状待ち行列」は, パーティション分割可能な並列計算機において, プロセススケジューリングのための待ち行列をパーティション毎に分散管理するものである. パーティションの管理プロセッサに待ち行列を分散配置することで, ボトルネックの問題を回避できる. 本稿では, この分散樹状待ち行列を用い, 負荷分散を考慮したタスクをパーティションに割り当てるアルゴリズム, およびタイムシェアリングに必要なラウンドロビンスケジューリング方式について提案し, シミュレーションの結果を示す. ここで提案されたプロセス管理方式は RWC プロジェクトで開発中の超並列計算機 RWC-1 上に実装される予定である.

A Process Scheduling Method for Massively Parallel OS

Atsushi HORI, Yutaka ISHIKAWA, Hiroki KONAKA,
Munenori MAEDA and Takashi TOMOKIYO

Real World Computing Partnership, Tsukuba Research Center

A process scheduling queue system for partitionable multi-computer system is proposed. We propose a new process scheduling queue system called "Distributed Queue Tree". In this system, process scheduling queues are distributed and locally managed on each partition. Thus, our process scheduling management is completely distributed and scalable. We also propose a load balancing mechanism and round robin scheduling on the queue system. These methods are evaluated with software simulations. The proposed queue system will be implemented on the massively paralle machine RWC-1 under developing.

1 はじめに

現在の商用並列計算機の多くは，柔軟な運用を可能とするため，プロセッサ空間を分割するパーティショニングが可能となっている（例えば，CM-5[2]）．これらのシステムでは，パーティションの選択はユーザがおこなう必要がある．バッチ処理における空きパーティションをどうやって見つけるかという研究（例えば，[1][3]）は多いが，時分割処理において負荷を分散させることを目的とした研究は数少ない．

プロセスマイグレーションを行なわない場合，プロセス切替オーバヘッドを無視すれば，時分割の場合でも計算機利用効率はバッチ処理の場合と同じである．しかし，時分割処理は応答性という側面において，逐次計算機の場合よりも効果があると考えられる．

一方，「超並列」を意識する上で最も重要なの事項のひとつは，処理の集中を避けることである．逐次型 OS あるいは共有メモリ型並列計算機の OS では，プロセスの走行待ち行列を集中管理することも可能である．しかしながら，大規模な疎結合型並列計算機においては，処理の集中は大きな問題となりうる．

本稿では，疎結合，パーティション分割可能な超並列計算機（例えば，RWC-1 [5]）において，文献 [6, 7] にあるような，包含関係にあるパーティションの時分割多重プロセス環境におけるプロセス管理の基本方式を提案する．

本稿で提案するプロセス管理方式は，タスク投入レベルでの負荷分散（あるいは，パーティションの自動選択）をおこなう．入れ子状にオーバラップしたパーティションは，時分割スケジューリングのタイムスロットに同期して，全体の（あるいは部分的な）パーティション分割の変更を可能とする．本方式は，基本的に分散かつ並列処理であるため，大規模な超並列疎結合型計算機に適用可能である．

本稿の構成

最初に想定する超並列計算機とプロセスについて示し，次に，本稿で提案する「分散樹状待ち行列」と，パーティション選択アルゴリズムおよびラウンドロビンスケジューリングについて説明する．最後に，これらの提案の基本動作を確認する意味でおこなったシミュレーションの結果を示す．

2 想定する超並列計算機

ここで対象とする超並列計算機は疎結合であり，以下のようにパーティション分割可能とする．

- それぞれのパーティションはオーバラップ可能である．
- パーティション分割そのものは静的に決定され，プロセスの実行状況に応じて動的にパーティションの大きさ／形状を変化させるようなことは考えない．
- オーバラップしたパーティションは時間的に排他的であり，任意の時刻においてすべてのプロセッサはひとつのパーティションにしか属さない．
- オーバラップしたパーティションは任意の時点で切替可能である．
- 複数のプロセスは，それらのプロセスがオーバラップしないパーティションに属する時，同時に走行可能である．

また，プロセスについては，以下のようなものを想定する．

- プロセスはひとつのパーティション内で並列に動作する．
- 全てのプロセスはプリエンプション可能である．
- ある時刻において，2 つ以上のプロセスが同じパーティションで走行することはない．
- プロセスは起動時にそのタスクが要求する大きさ（プロセッサ数）と同じ大きさのパーティションが割り当てられる．割り当てられたパーティションはプロセスの終了まで変わらない（プロセスマイグレーションは考えない）．

3 分散樹状待ち行列

逐次あるいは共有メモリ型計算機においては，プロセス待ち行列を集中管理する方式が採用されている．プロセスの待ち行列は，プロセスの状態遷移に伴い，待ち行列を頻繁に出入りする．このため，疎結合でパーティション分割される並列計算機の場合，待ち行列管理がボトルネック

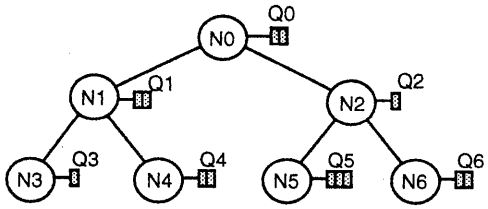


図 1: 分散樹状待ち行列の例

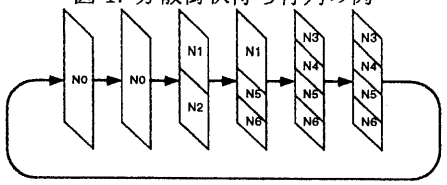


図 2: ラウンドロビンスケジューリングの例

になり適切ではないと考えられる. ボトルネックを避けるには, オーバラップしたパーティションに対応し, それぞれのパーティションにスケジューラを設ける分散管理方式が望まれる.

そこで図 1に示すような待ち行列を樹状に展開した形を考える. それぞれの節はパーティションに 1 対 1 で対応している. 全体のプロセッサ数を 8 (2^3) とすれば, レベル 0 は 8 プロセッサ, レベル 1 は 4 プロセッサ, レベル 2 は 2 プロセッサ, レベル 3 は 1 プロセッサのパーティションに対応する. 待ち行列は, それぞれのノード毎に分散して存在する (図中, ノードを表す○の右横にあるのが待ち行列を示す). このように待ち行列をパーティションに対応して分散管理することで, プロセスの状態遷移に伴う待ち行列からの出入りは, パーティションの内部で局所的に処理可能となる.

図 2は, ラウンドロビンスケジューリングの例である. この図にあるように, パーティション分割は時分割スケジューリングにおいて, タイムスロット毎に変化する. 図 1 および図 2 では, 2 分木を用いたが, 以下の議論では n 分木 ($n \geq 2$) でも基本的に同じである.

このような分散樹状待ち行列では, (1) タスクをどのパーティションに投入すべきか (長期スケジューリング), (2) 時分割処理に必要なラウンドロビンスケジューリングをどうやって行なうか, という 2 点が新たな問題となる. 以下, これらの問題に対処するための方式について検討する.

3.1 用語の定義

- 分散樹状待ち行列 (Distributed Queue Tree)
  図 1にあるような, 樹状に展開された待ち行列. 木の深さ方向 (根あるいは枝に向かう方向) に対して排他的であり, この方向に走行しているプロセスは高々ひとつしか存在しない.

- 部分待ち行列長 (Subqueue-length)
  ある部分木において, 自分自身のノードに存在する待ち行列長と, 直下の子ノードの部分待ち行列長の最大値を加えたもの. ノードが葉の場合は, そのノードの待ち行列長となる. この値は, そのノード以下の部分木に含まれるプロセスが, 少なくともその部分待ち行列長に 1 回以上スケジューリングされることを意味する. ルートノードにおける部分待ち行列長を「システム全体の待ち行列長」と呼ぶ.

- プロセッサ割当量 (Assigned Processor Amount)
  あるノードのプロセッサ割当量は, 直下の子ノードのプロセッサ割当量の総和に, 自ノードの担当するパーティションの大きさと自ノードの待ち行列長 (部分待ち行列長ではない) の積を加えたものである. プロセッサ割当量は, そのノードが必要としているプロセッサ数を示す.

- タスク (Task)
  仕事 (ジョブ). タスクは, 処理に必要なパーティションの大きさ (タスクサイズ) と完了するまでに要する時間 (タスク長) のふたつの属性を持つ. タスク長は他にタスクが存在しない場合の処理時間であり, タスク長より短い時間でタスクが終了することはない. あるタスクの仕事量は, タスクサイズとタスク長の積で表される.

- プロセス (Process)
  タスクの処理.

3.2 パーティションの選択

単一待ち行列管理と異なり, 樹状待ち行列では, 与えられたタスクをどのパーティションに割り当てるべきかを判断する必要がある. 判断する基準としては, (1) プロセッサの全体での利

用率, (2) スケジューリングの公平性などが考えられる. ここでは, タスクの属性として必要なパーティションの大きさを持つものとし, 以下に示すようなパーティション選択アルゴリズムを考える. このアルゴリズムにおいて、タスクに関する情報は必要なパーティションサイズしか与えられていないものとする.

タスクの投入はすべてルートノードから始められる.

1. 受けとったタスクのサイズが, 自ノードの担当するパーティションの大きさと同じであれば, 自ノードの待ち行列に追加する.
2. そうでない場合は, 子ノードの部分待ち行列長のもっとも短いノードにタスクを投げる. 部分待ち行列長が同じ場合は, 最もプロセッサ割当量が少ないノードにタスクを投げる (以下, 再帰処理) [1].

このアルゴリズムにおいて, 部分待ち行列長を重視したのは, 公平性を保つためである. また, 部分待ち行列長が同じ場合には, プロセッサ割当量の少ないノードの方がパーティションに空きが生じている可能性が高いと思われるからである.

素直に考えれば, プロセッサ割当量のようなものよりもビットマップなどを用いてパーティションの空きを表現し, Best-Fit や First-Fit 方式を採用する方式も考えられる. しかし, これらの方式では, タスクをパーティションに割り当てる時点では最適かもしれないが, 以後のタスク投入パターンによって最適な状態が崩れる可能性が否定できない. このため, いたずらに複雑なアルゴリズムを用いずに, より単純な「プロセッサ割当量」を基準とした方式とした.

図3に例を示す. 図中, ノード内の Q_L はそのノードの部分待ち行列長を示し, P_T はプロセッサ割当量を示す. シェーディングされたノードは, そのノードの待ち行列が空でないことを示し, 太丸のノードは, 前の状態 ((a)→(b)→ (c) の順) から変化があったノードを強調している.

(a) の状態の時に, サイズ2のタスクが投入され, 先のアルゴリズムに従って, パーティションが選択された結果を (b) に示す. 同様に, (b) の状態の時に再度サイズ2のタスクが投入された結果の状態を (c) に示す.

[1] シミュレーションによれば, プロセッサ割当量も同じであった場合, 前回のタスクを投入したノードと反対のノードに割り付ける方式が, 常に同じ側のノードに割り付ける場合よりも効果のあることが認められている.

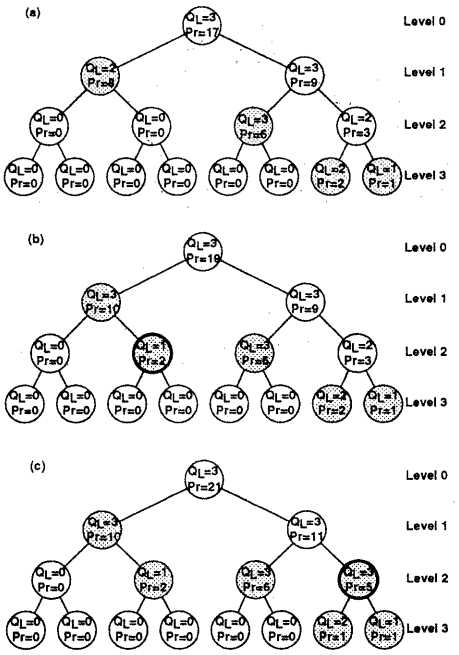


図 3: 負荷分散の例

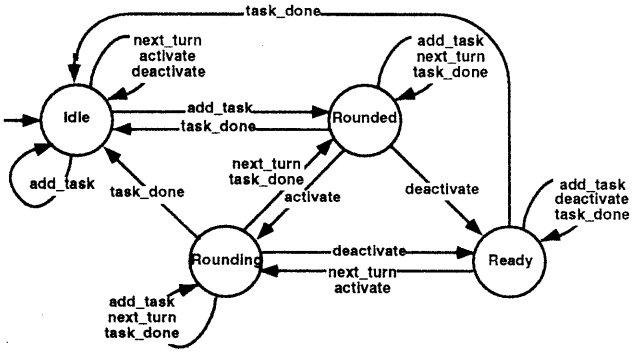
3.3 ラウンドロビンスケジューリング

ここでは, 時分割を実現するための分散樹状待ち行列のスケジューリングプロトコルを考える. 各ノードは, 互いにメッセージ通信を交わしながらスケジューリングをおこなう. 以下に, ラウンドロビンスケジューリングと上記のアルゴリズムに基づいたパーティション割当に必要な通信メッセージの種類と意味について説明する.

図4にノードの状態遷移を示す. この図において, タスクの終了を示す task_done メッセージが, プロセスを走らせていないはずの状態である "Rounded" や "Ready" 状態でも発生していることに注意を要する. これは, ノードの状態遷移がプロセスの状態遷移と非同期におこなわれるからである.

以下のメッセージの説明において, ↓ はルートから下位に向かって流れるメッセージを示し, ↑ は葉からルートに向かって流れるメッセージを示す.

- add_task (↓)
  新規タスクの投入. タスクサイズが自ノードのパーティションの大きさを等しい場合



状態の説明

Idle：自ノードの待ち行列が空である状態。
Rounded：タスクが投入されてまだスケジューリングされていない状態、または自ノードが管理する待ち行列のラウンドロビンスケジューリングが1周した状態を示す。この状態におけるnext_turnメッセージはそのまま下位ノードに転送される。
Rounding：ラウンドロビンの最中である状態。
Ready：スケジューリング待ちの状態。

図 4: ノードの状態遷移

は, 自ノードの待ち行列に投入し, そうでない場合は, 前述したアルゴリズムに従って下位ノードに転送する.

- next_turn (↓)
  ラウンドロビンにおけるプロセス切替をおこなう. このメッセージを受け, 自ノードの待ち行列の次のプロセスをスケジューリングする. もし, 待ち行列が空であったり, 待ち行列の最後尾のプロセスを実行中であった場合は, 自ノードの状態を "Rounded" とし, next_turn メッセージを下位ノードに転送する.

- rounded (↑)
  ラウンドロビンが1周したことを親ノードに通知する. このメッセージは自ノードが "Rounded" 状態にあり, 全ての下位ノードから rounded メッセージを受けとった場合に発生する. このメッセージをもって, このノード以下のプロセスが最低1回はスケジューリングされたことを示す.

- activate (↓)
  ノードを活性化する（待ち行列が空でなければ, Rounding 状態にする）. このメッセージはルートノードが直下の全てのノードより rounded メッセージを受けとった場合（新しいラウンドに入ることを意味する）, および, 下位の子ノードの rounded メッセージの同期待ちの時に, より早く rounded を報告した子ノードを再度周回させる場合に発生する.

- deactivate (↓)
  ノードを不活性化する（待ち行列が空でなければ, Ready 状態にする）. activate メッセージを受けて活性化したノードが, 全ての下位ノードを不活性化する時に発生する.

- task_done
  タスクの終了. このメッセージはノード内で発生する. 実際には, プロセスが中断された場合のように待ち行列から外される場合にも対応する.

- proc_amount, subq_length (↑)
  タスク投入／終了に伴い, 自ノードの状態変化（部分待ち行列長およびプロセッサ割当量）を上位ノードに通知する.

このラウンドロビンスケジューリングについて, もう少し分かり易く説明する. 樹状待ち行列の図において, アクティブなノード（状態遷移で "Rounding" 状態にあるノードのこと）を結んだ曲線を「前線（front）」と呼ぶことにする. 図5に前線移動の例を示す. 図中の○中の長方形は待ち行列のエントリを示し, 前線は木を横断し左右に伸びる曲線で示した. 前線より上のノードの状態は全て "Rounded" であり, 前線より下のノードの状態は全て "Ready" である. 前線は葉に到達するまでは, 常に根から葉に向かう.

同じレベルのノードが全て同じ部分待ち行列長であるような場合, 前線は水平線となり（図中の t0, t1 の前線）, 全ての葉に到達した時点でラウンドロビンが1周したことになる. しかし, 図 5に示すように, 部分待ち行列長が不揃いの場合は, 部分木がラウンドロビンにおける「1周」するまで, 先に周回した方の枝を繰り返

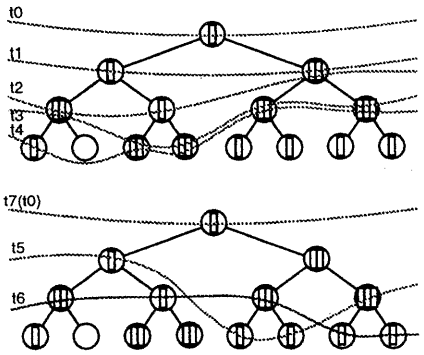


図 5: 前線移動の様子の例

し周回させる．こうして，全ての枝が1周するまで，前線は部分的に上下に波を打つ挙動を見せる（図中の t3～t6 の前線）．

4 シミュレーション

これまで述べてきた「パーティション選択アルゴリズム」と「分散樹状待ち行列におけるラウンドロビンスケジューリング」の基本動作を理解する意味で，シミュレーションをおこなった．

4.1 シミュレーション条件

基本となるシミュレーションは，以下に示す条件でおこなわれた．

1. 分散樹状待ち行列は2分木とした[2]．それぞれのパーティションはすき間のない入れ子にある．
2. パーティションのサイズは，2^n（$n = 0, 1, ...6, 7$）とする[3]．
3. タスクサイズは，2^n（$n = 0, 1, ...5, 6$）とし，指数分布（大きなタスク程，投入頻度が少ない）とする[4]（表1）．

[2]これは，RWC-1 のネットワークトポロジー [4] の性格から，1次元でパーティション分割した方がよりよい性能を示すからである．

[3]RWC-1 は最大 1,024 PE 構成であるが，最小パーティションサイズを8とした場合を考慮したため．

[4]タスクサイズにおいて 128 の大きさはシミュレーションの対象外とした．これは最大構成と同じ大きさのタスクサイズは，単一待ち行列と同じ結果を招くため．

表 1: 投入したタスクサイズの分布

サイズ	1	2	4	8	16	32	64
頻度	1,904	948	474	237	119	60	60

4. タスクの長さは 100～16,483 単位時間の範囲で一様乱数とする．
5. 1単位時間を時分割における量子時間とする．

シミュレーションは，負荷を変えて，全部で3ケースおこなった．それぞれの条件を表2に示す．各々において，待ち行列がある程度定常状態になるのを待つため，シミュレーション時間の1／6に相当する最初の部分は計測をおこなっていない（ウォーミングアップ）．タスクの投入は一定時間間隔でおこなった（表中 Interval）．この表における「負荷率」とは，投入したタスク仕事量の総和をシミュレーション時間（表中 “Sim. Time”）とプロセッサ台数の積（128）で割ったものである．ケース1は，負荷率が1より小さいと同時に1に近いように選び，ケース2，3は，ケース1のシミュレーション時間とタスク投入間隔をそれぞれ2倍，3倍したものである．このため、ケース2，3の負荷率はケース1の 1/2, 1/3 になる．タスクの投入パターンを全く同じにすることで，ケース毎の比較が容易になる．

なお，シミュレーションにおいて，スケジューリングに伴うオーバヘッドは考慮されていない．また，プロセスは実行途中で I/O 待ちなどの中断はしないものとした．

表 2: シミュレーションの条件

Case	Sim. Time	Interval	負荷率
1	1,000,000	263	0.9885
2	2,000,000	526	0.4942
3	3,000,000	789	0.3295

4.2 シミュレーション結果

それぞれのケースにおける最大待ち行列長（表中 QL-MAX），実績計算量（与えられたタスクに対し実際に処理した仕事量），およびプロ

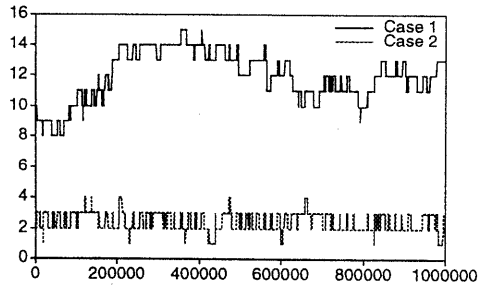


図 6: 待ち行列長の時間推移（ケース１＆２）

セッサの利用率（実績計算量を，プロセッサ数とシミュレーション時間を掛けた値で割ったもの）を示す．

表 3: シミュレーション結果

Case	QL-MAX	実績計算量	利用率
1	15	1.232e+08	0.9631
2	4	1.259e+08	0.4919
3	3	1.261e+08	0.3284

ケース１とケース２における全体での待ち行列長の時間推移を図 6に示す（ケース２における時間はケース１と対応させるため 1/2 にしてある）．このグラフはタスク投入直後に全体の待ち行列長をサンプリングした結果である．この図を見る限り，高負荷状態においてスケジューリングに不安定な挙動は認められていない．

図 7, 8, 9はそれぞれのケースにおいて，タスクサイズと実実行時間比率（タスクの実実行時間をタスク長で割ったもの．この図では小数点以下切捨）の頻度分布を示したものである．これらのグラフから，システムの負荷が減少するに従い，タスクの実実行時間比は順当に減少することが分かる．

図 10に，タスク終了時点でサンプリングしたタスクの実実行時間比（縦軸）の時間推移を示した．参考のため，この図に待ち行列長を重ねてプロット（図中、実線で示す）した．この図から，タスク間で実実行時間比に最大 6 倍程度の格差が生じている（横軸 700,000 付近）ことが分かる．これは，タスクの終了により生じた部分待ち行列長のアンバランスが，その後のタスク投入により解消できないためと考えられる．この現象は下位ノード（小さいタスク）程，顕

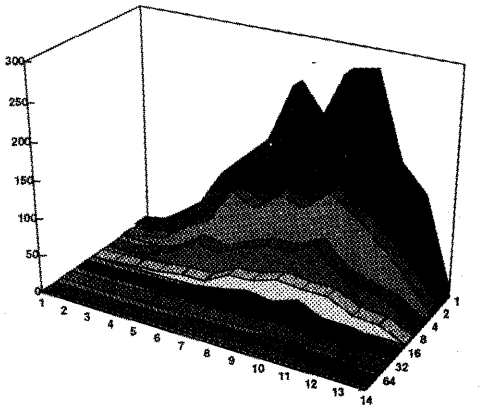


図 7: タスクサイズと実実行時間比（ケース１）

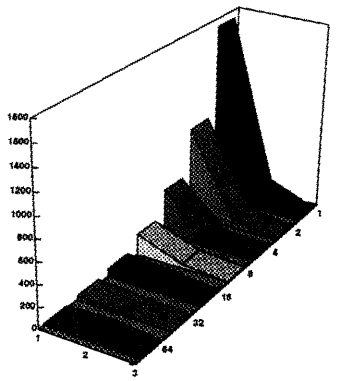


図 8: タスクサイズと実実行時間比（ケース２）

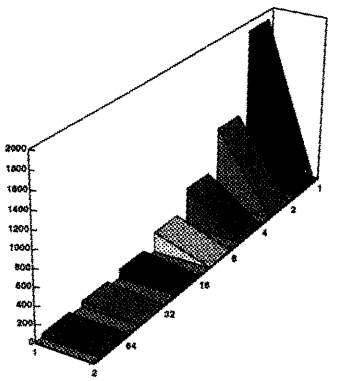


図 9: タスクサイズと実実行時間比（ケース３）

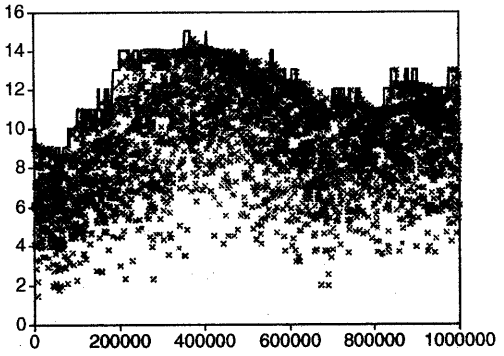


図 10: 実実行時間比の推移（ケース 1）

著に現れるものと考えられる.

まとめ

オーバラップ可能なパーティション分割プロセスのスケジューリング方式について「分散樹状待ち行列」を提案した. 同時に, この枠組において, タスクが要求するサイズのパーティションを選択するアルゴリズムと, 時分割処理に必要なラウンドロビンスケジューリング方式を示した. これらの方式は, シミュレーションにより基本的な動作が確認されたと考えられる.

しかしながら, スケジューリング上の公平さに問題があること, 実際的な時分割スケジューリングにおいて優先度が必要なことなど, 課題も残されている. 今後の研究の方向としては以下のものを考えている.

1. パーティション選択アルゴリズムにおいて, First-Fit など別なアルゴリズムを用いたシミュレーションをおこない, 比較・改良をおこなう.
2. より実用的なスケジューリングを実現するため, 本稿で提案した分散樹状待ち行列のラウンドロビンプロトコルに優先度を導入する.
3. プロセスマイグレーションなどにより, タスクの実実行時間比の格差を解消する方式を検討する.

本稿で提案した分散待ち行列によるプロセススケジューリング方式は, RWC-1 のオペレーティングシステムに実装する予定である.

謝辞

RWC 超並列ソフトウェアワークショップに参加の各位からは数々の貴重なアドバイスを頂いた. ここに感謝の意を表す.

参考文献

[1] John K. Ousterhout. Scheduling Techniques for Concurrent Systems. In *Proceedings of Third International Conference on Distributed Computing Systems*, pp. 22–30, 1982.

[2] Thinking Machines Corporation. *Connection Machine CM-5 Technical Summary*, November 1992.

[3] Yahui Zhu. Efficient Processor Allocation Strategies for Mesh-Connected Parallel Computers. *Journal of Parallel and Distributed Computing*, Vol. 16, pp. 328–337, 1992.

[4] 横田隆史, 松岡浩司, 岡本一晃, 廣野英雄, 堀敦史, 児玉祐悦, 佐藤三久, 坂井修一. 超並列計算機 RWC-1 の相互結合網. 情報処理学会研究報告 ARC, pp. 25–32, August 1993.

[5] 坂井修一, 岡本一晃, 松岡浩司, 廣野英雄, 児玉祐悦, 佐藤三久, 横田隆史. 超並列計算機 RWC-1 の基本構想. 並列処理シンポジウム JSPP'93, pp. 87–94, 1993.

[6] 堀, 石川, 坂井, 小中, 前田, 友清, 松岡, 岡本, 廣野, 横田. 並列計算機オペレーティングシステムカーネル SCore におけるプロセス管理とハードウェア支援機能. コンピュータシステム・シンポジウム論文集, pp. 59–66. 情報処理学会, October 1993.

[7] 堀, 石川, 小中, 前田, 友清. 超並列システムカーネル SCore の構想. システムソフトウェアとオペレーティング・システム研究会資料, pp. 57–64. 情報処理学会, August 1993.