TM-1169

性能管理 OS における動的負荷分散方式とその実現

前田　宗則、神田　陽治（富士通研究所）

April, 1992

ICOT

Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan

(03)3456-3191～5
Telex ICOT J32964

Institute for New Generation Computer Technology

性能管理OSにおける動的負荷分散方式とその実現

前田 宗則(こうだ), 神田 陽治

株式会社富士通研究所 国際情報社会科学研究所

e-mail: m-maeda, kohda@iias.fujitsu.co.jp

マルチタスク処理をサポートする並列マシン用OSを構築する上で重要な課題の一つは負荷分散である．我々は，負荷分散に焦点を絞った並列マシン用OSの開発を進めている．本OSでは，アプリケーションプログラムから負荷分散に関する記述を分離しOSの管理化に置くことで，アプリケーションのポータビリティを向上させるだけでなく，OSによる負荷分散戦略の自動立案への可能性が開かれる．本報告では，OSの概要を述べた後プログラムとプロセスの動的再配置に基づく新しい動的負荷分散法とその実現について検討する．さらに並列マシンmultiPSI上に構築されたOSプロトタイプの実行例を示す．

Dynamic load balancing and its implementation on Multi-Task Mix OS

Munenori Maeda, Youji Kohda

Fujitsu Laboratories, IIAS

17-25, Shinkamata 1-chome, Ohta-ku, Tokyo 144, Japan

When OS supports multitasking on a parallel machine with many processors, load balancing between processors is a severe problem. To tackle this problem, we proposed to make program and load balancing strategy separated from user's application. It brings the high portability of the program, moreover automatic planning of an optimal strategy will become possible when it's managed by OS. In this paper we show our OS overview and discuss about a new dynamic load balancing method and its impremention which is based on the migration of process and program segment. Execution examples of the OS prototype implemented on multiPSI are also shown.

1 はじめに

我々は，性能管理 OS(別称:Multi-Task Mix OS)[1] と呼ぶ汎用並列マシン用オペレーティングシステムの開発を進めている．現状では，multiPSI[8], PIM[6] といった疎結合並列マシンと並列論理型言語 KL1[9] のアプリケーションをターゲットとしている．本報告では，性能管理 OS の提供する機能の1つである，プロセッサの負荷分散の実現方式とその実現について考察する．

まず本稿で用いている用語について述べておく．ジョブとは，利用者が投入するいわゆるアプリケーションのことである．

タスクとは，ジョブに負荷分散のための情報（戦略）を加えたものである．負荷分散戦略は，どういった負荷分散方式を採用するかによって記述の内容，方法ともに異なるが，おおまかには，利用可能な複数のプロセッサがある場合に，どういったシチュエーションで何台の（あるいは，どの）プロセッサにいかにジョブを分割して与えるかを指定するものである．

ジョブの処理速度（スループット，ターンアラウンドタイム）を向上させるためには，プロセッサの稼働率を上げることが必要である．プロセッサの稼働率を上げるためには，プロセッサの負荷分散を適切に行うことが重要になる．

負荷分散は，静的負荷分散方式と動的負荷分散方式に大別される．一般に，ジョブの振る舞いを予め見積もることが難しいので，静的負荷分散は適用可能なアプリケーションが限られる．それゆえ効率のよい効果的な動的負荷分散が望まれる．

これまでに提案された疎結合並列マシン用の動的負荷分散方式の多くは，負荷分散戦略をアプリケーションプログラムに組み込むものである．負荷の高い（低い）プロセッサが，OS もしくは自前の負荷分散機構に負荷の低い（高い）プロセッサを問い合わせ，そのプロセッサに（そのプロセッサから）負荷の一部を移動することをプログラム中に陽に記述する．こういった負荷分散戦略をプログラムに組み込んでチューニングする方法では，性能（処理速度）の大幅な向上をもたらすが，チューニングが適切でないなら性能は極端に低下することがある．これは，負荷分散処理にかかるコストに加え，疎結合マシンではプロセッサ間の通信コストが大きいからである．チューニング作業は，ユーザーに大きな負担を与えている．あるいは OS がマルチタスク処理を行う場合を考えてみる．この場合タスク間に相互干渉が存在するために，単一のジョブを繰り返し実行してチューニングされた負荷分散戦略が必ずしも有効ではなくなる．

そこで我々が目指すのは，“チューニングしなくともまあまあの処理速度を与える”負荷分散機構を持つ OS の構築である．“まあまあの”という言葉は曖昧であるが，これはユーザーによって十分に最適化された負荷分散戦略を持つタスクの処理速度を上回ることはないという程度の意味で用いている．

本論に入る前にその構成を明らかにする．まず2節で性能管理 OS の概要を示す．次に 3節で OS の提供する2つの負荷分散機構について述べる．4節では動的負荷分散機構の実現技法を提示する．性能管理 OS の実行例は 5節で示される．最後はまとめである．

2 性能管理 OS の概要

前節で述べた通り，性能のチューニング作業は，ユーザーに大きな負担を与えており，また，複数のタスクが次々投入されるマルチタスク処理では，シングルタスクの環境で得られた最適戦略の有効性が保証できないという問題がある．将来，安価な並列マシンが現在のワークステーションのように普及する状況が訪れた時，ユーザーは一つ一つのアプリケーションをチューニングする手間を惜しむことはないのだろうか．

これに対する性能管理 OS のアプローチは次のとおりである．まず従来のようにプログラム中に負荷分散戦略の記述を行わない．すなわち，アプリケーションからプログラムと負荷分散戦略を分離するのである．分離された負荷分散戦略は，OS によって管理されチューニングの対象となる．チューニングの内容は以下のような作業である．

- アプリケーションプログラムに割り付けられるべきプロセッサ台数とルーティングの決定

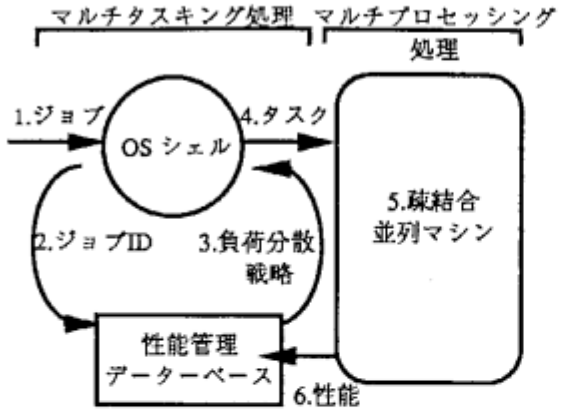

図 1: Multi-Task Mix OS Overview

- 割り付けられたプロセッサに均等に負荷を分散するようなパラメータとデータ構造の調整

前者は，アプリケーションごとに効率よく分散できるプロセッサ台数の範囲があるという仮定に基づいている．1プロセッサで十分実行できる規模の小さいジョブを64プロセッサに負荷を分散させる必要はないのである．後者は，3.2節で述べるマルチプロセッシング処理で詳しく述べる．

性能管理OSは，各ジョブに適用された負荷分散戦略とそのときの性能を記録する性能管理データベースを持ち，ジョブが投入されると，性能管理データベースからこれまでの実行履歴を参照し，状況に適した負荷分散戦略を立案する．得られた負荷分散戦略とジョブは，タスクとして並列マシン上で実行される．実行後，性能と負荷分散戦略は性能管理データベースに登録され次にそのジョブを実行する際に利用される．これら一連の処理をマルチタスキング処理ということにする．また，並列マシン上でのタスクの制御をマルチプロセッシング処理という．以上を模式的に表したのが図1である．

ジョブのチューニングは時間に沿って行われるのみでなく，並列マシン上の余力のあるプロセッサを用いて空間的にも行われる．すなわち，ジョブに与える負荷分散戦略とは異なった戦略を持つ複数のタスクを余力のあるプロセッサに割り付けて同時に実行するのである．これを実験タスクと呼び，本来のタスクを本タスクと呼ぶことにする．実験タスクは次のような場合に起動される．

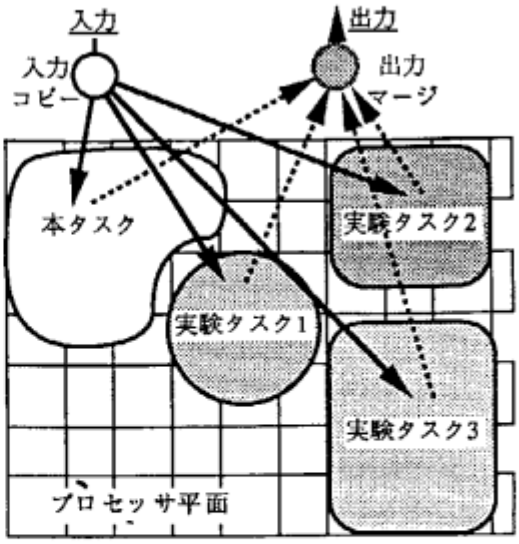

図 2: 実験タスクを併用した本タスクの実行

- 本タスクを実行する際に，1つ以上の実験タスクを実行するのに十分な台数のプロセッサがアイドル状態にある．この場合，投入された実験タスクの数が予め決められた規定値に達するか，アイドルプロセッサを消費しつくすまで実験タスクをフォークする．各実験タスクには，本タスクへの入力データをコピーして与える．また実験タスクからの出力はマージされ，最も早く与えられた出力を優先するとともに，出力内容が重複することを防ぐ．最も早くデータを出力したタスク識別子が性能管理データベースに保存される．これを図2に示す．

- ジョブ投入のタイミングとは独立に，適当なインターバル時間でアイドル状態にあるプロセッサを調べる．その結果，十分な台数のプロセッサがアイドル状態にあることが判明した場合，利用頻度が高いか，最適化に関して高い優先度を持つジョブに対して実験タスクを自動的に起動する．この場合本タスクは存在しない．各実験タスクには過去にユーザが与えた入力を再現して与え，実験タスクからの出力は放棄する．

複数の実験タスクの実行によって，新たに投入されるジョブの実行が遅らされてはならない．それゆえジョブ投入時に必要な台数のアイドルプロセッサが確保できず，かつ実験タスクが存在しているなら

ば，いくつかの実験タスクの実行を中断あるいは放棄して使用中のプロセッサをOSに開放する．これらのプロセッサは新しいジョブに利用される．

これまでに述べたように性能管理OSでは，利用可能な複数のプロセッサにどのタスクを割り付けるかということと，あるタスクに割り付けられたプロセッサ間で動的にいかに負荷を均等化するかということが主要な課題となる．前者はマルチタスキング処理における静的負荷分散問題に，また後者はマルチプロセッシング処理における動的負荷分散問題にそれぞれ帰着される．次節では，これらの負荷分散問題について考察する．

3 負荷分散機構とその処理

3.1 マルチタスキング処理と静的負荷分散

各タスク毎に割り付けられるべきプロセッサは，要素を$PN_i(i \geq 0)$とする有限の列で指定される．ここで，添字iは論理プロセッサの識別番号であり，PN_iは物理プロセッサの識別番号である．列には重複する要素が含まれていることも許す．この列を論理マシン記述と呼ぶことにする．

性能管理OSの用いるタスクの物理プロセッサへの割り付け方針は次の通りである．

- タスクに対して十分な台数のプロセッサが存在する場合は，物理プロセッサがオーバーラップしないように論理マシンを決定する．
- 物理プロセッサの台数には制限があり，投入されるタスクの数は，物理プロセッサの台数を上回ることがある．各タスクの平均的な負荷は性能管理データベースより見積もりことができるので，静的に負荷最小の物理プロセッサを検索しそれを起点にして必要な台数の論理プロセッサを確保する．

以上の方針によって各プロセッサの平均稼働率をできるだけ一定にする．論理プロセッサと物理プロセッサの対応及び物理プロセッサの確保を図3に示す．

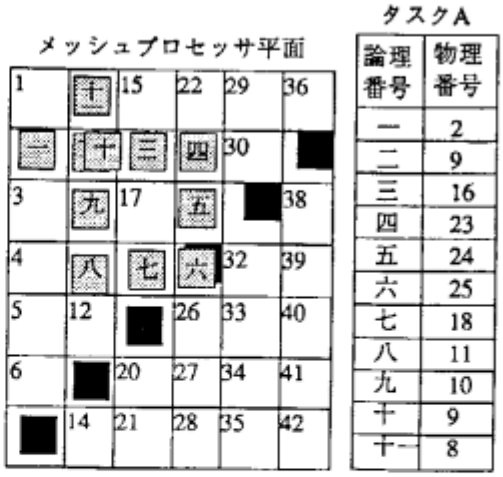

図 3: 論理プロセッサから物理プロセッサへのマッピング

3.2 マルチプロセッシング処理と動的負荷分散

従来の動的負荷分散方式[1]では，プログラム（実行コード）は実行に先立って各プロセッサの主記憶上にロードされているということを仮定している．これを2つの例を挙げて吟味してみる．

- 負荷分散処理によってあるプロセッサに割り付けられたプロセス，及びそのプロセスから生成される全てのサブプロセスが，プログラム中の全ての手続きにアクセスすることは殆どない．
- 仮想記憶は，ページングアルゴリズムにより必要なページもしくはプログラムセグメントを二次記憶から主記憶に動的に移動し，さしあたって必要でないプログラムセグメントを主記憶から削除するものである．これは主記憶上のプログラムが動的に変更されるように見える．

これらを踏まえて，“タスクに割り付けられたプロセッサが全て同じプログラムを内蔵する必要はなく，プロセスと対応するプログラムが，あるプロセッサで揃えば実行することが可能である”といえる．

[1]KL1を対象とする動的負荷分散方式では，プロセッサの実行待ち行列に存在するプロセス数をプロセッサ間で均衡させることに主眼をおく．これは，プロセス（あるいはゴールともいう）の並列粒度が十分小さく，論理的な取り扱いも容易であるからである．

我々は，タスク実行時にプログラムセグメントをプロセッサ間で移動させることで間接的にプロセスを分散させる方針をとる．具体的な手続きを挙げると以下のようになる．

1. 各論理プロセッサは，適当なインターバルタイムでスケジューリングキューに存在するプロセス数（負荷）を調べる．

2. 負荷が高いと判断されたならば，プロセッサ上にあるプログラムセグメントの内の一つをある基準の従って選出する．→ 5.

3. 負荷が低いと判断されたならば，次の識別番号を持つ論理プロセッサにプログラムセグメントとそれを参照するプロセスの転送依頼を行う．

4. 負荷は適切な範囲である．→ 1.

5. 選ばれたプログラムセグメントの転送方法には，duplicate転送とmove転送の2通りがある．duplicate転送は，プログラムセグメントの複写を作成しそれを転送するものである．この場合には，元のプロセッサには依然としてそのプログラムセグメントが存在することになる．また，move転送では，元のプロセッサからプログラムセグメントを消去する．
   転送方法に関わらず，プログラムセグメント転送と同時にそれを参照するいくつかのプロセスも同時に移動する．→ 1.
   転送方法は，各プログラムセグメントの持つ属性としてタスク投入前に決定される．

プログラムのセグメント分割パターン，各プログラムセグメントの転送属性がタスクの性能を決定する主因であり，OSによってチューニングされる．

4 動的負荷分散機構の実現

我々は，異なる2つのKL1処理系: Unix上のPDSS及びmultiPSI上のPIMOSを開発環境として用い，性能管理OSのプロトタイプを開発した．性能管理OSは，視覚的ユーザーインターフェース，マルチタスキングシェル，マルチプロセッシングシミュレータの3つのサブシステムから構成されており，視覚的ユーザーインターフェースシステムの一部を除き全てKL1で記述されている．本節では，マルチプロセッシングシミュレータ(MPS)に焦点を当ててKL1上での実現手法を述べる．

MPSはKL1メタインタプリタでモデル化できる．KL1メタインタプリタの基本動作は，次なる処理の繰り返しである: “まずスケジューリングキューからプロセスを取り出し，それをリダクションする．その結果得られたサブプロセスをスケジューリングキューに格納する”．

MPSではメタインタプリタを以下のような点で拡張する．

1. 1台の論理プロセッサをメタインタプリタで表現し，論理マシンは互いにストリームで通信する複数のメタインタプリタとして実現する．

2. プロセス及びプログラムの転送処理の記述を加える．プログラムの転送処理を実現するためには，実行プログラムを陽に扱いそれが格納されているワークスペースを明示しなければならない．

3. あるプロセスのリダクション処理中にそれが参照しているプログラムセグメントの転送が生じたときは，リダクション処理を強制放棄させてプロセスをスケジューリングキューに再格納することが必要である．

1.は特に問題なく実現できる．2., 3.の拡張は実行効率に大きな影響を与えるので十分な検討が必要である．以下ではそれぞれについて実現上の問題点を挙げ，プロトタイプで採用した技法を述べる．

4.1 プログラムセグメントの編集処理

KL1には，実行時にプログラムセグメントを付加したり削除したりする機能がない．プログラムセグメントの集合を編集する単純な方式では，プログラムセグメントをいわゆるフローズンフォームと呼ばれる基底項で表現しその集合をデータとして保持する．この方法ではリダクション処理機構を自前で用意しなければならないので効率が悪い．そこで，

実際にプログラムを動的に移動するのではなく，プログラムの特定のセグメントのみアクセスできるようなマスク機構とマスクパターンを用意することでシミュレートすることにした．

まずユーザープログラムに次なる3つの前処理を施す．

1. 各プログラムセグメントにユニークな識別番号を割り当てる．
2. 各ホーン節にマスクパターンが具体化する引数 P を加える．
3. 各節のガード部に，P を参照して自身のプログラムセグメントが存在するかどうかチェックし，存在しないならばこのホーン節がコミットされないようにする手続きを挿入する．

その上で実行時にプロセスの引数にプログラムセグメントのマスクパターンを設定する．

以上の操作を効率よく行なうために，各プログラムセグメントには，2の巾乗の整数を割り当てた．その結果プログラムセグメントの存在のチェックは，単純なビット演算を用いて高速に行なうことが可能になった．マスクパターンの対応するプログラムセグメントのビットをON/OFFすることで，プログラムセグメントの集合を付加したり，削除することが容易にシミュレートできる．

4.2 リダクション処理の強制放棄

リダクション処理（特にガードの述語の実行）をKL1処理系に任せるか否かで実現の容易さも効率も大きく異なる．KL1処理系に任せないならば，変数の管理機構[2, 3]を用意することになる．変数管理はかなり負担の大きい処置である．実装方法にも依存するが，単一化操作で数倍から数十倍遅くなると報告されている[2]．しかしながら変数の束縛検査を陽に行うことができるため，プロセスがサスペンドしているか否かを調べることができる．

我々の採用した方法は，リダクション処理をKL1処理系に委任するものである．この方法は効率がよい反面，変数の束縛検査を行うことができない．それゆえサスペンドしているプロセスの回収処理が難しい．具体的なリダクション処理放棄の技法は次の通りである．

全てのプロセスの手続きに処理の強制放棄のための引数 C を加える．KL1では独立なOR節で構成されている処理の本体に，新たに C が具体化した場合の処理を記述した節を追加する．C は，強制放棄が決定したと同時に定数に具体化される．この節の選択優先度が他のOR節より高いことが保証されるならば，入力の具体化待ちでサスペンドしている場合にも実行放棄が行える．実行放棄されたプロセスは，スケジューリングキューに回収される．

この技法では，いつプロセスがスケジューリングキューに回収されるかまで知ることができない．単一環境で動作するPDSSでは即座に回収されるが，multiPSI上では多少遅延が生じ，処理が幾分不安定になることが判明している．しかしながら効率の点から現在は本技法を採用している．

5 性能管理OSプロトタイプの実行例

図4に性能管理OSの表示画面を示す．視覚的ユーザーインターフェースにより多くの作業がマウスによりメニューを選択するだけの操作で行える．表示画面の左上に表示されてるのは，4 ⋆ 4の物理プロセッサから構成される並列マシンと現在実行されている複数のタスクである．各タスクは異なるハッチパターンで表示されている．また，物理プロセッサは，3 ⋆ 3の論理プロセッサに分割されてタスクに割り付けられている．画面左下には，各タスクの名称とそれに与えられている負荷分散戦略がテキストで分かりやすく表示されている．

画面の右上に表示されているのは，マルチタスキング実験実行のためのメニューである．マニュアルモードにおいては，ジョブと負荷分散戦略をそれぞれ選択した上で並列マシンに投入することができる．また，オートマチックモードにおいては，任意のジョブを与えるとOSが負荷分散戦略を立案し並列マシンに投入する．図は，ジョブprime(500,x)について複数の実験タスクが起動している状況でオートマチックモードでジョブqueen(6,x)を指定しているスナップショットである．

次にジョブprime(100,x)において，与えられた2つの負荷分散戦略で性能がどの程度異なるかについて計測した結果を図5に示す．2つの負荷分散戦略は，全てのプログラムセグメントにmove属性を与えたものとmove及びduplicate属性を最適に与えたものである．公平に評価するため，物理プロセッサと論理プロセッサを1対1に対応させ，共に4台の物理マシン($No.3 \sim No.6$)を割り付けた．

図は，横軸に実行時間を縦軸にプロセッサの識別番号をとったものである．ハッチパターンは明度が低い程プロセッサの稼働率が高いことを示している．図左は，戦略を全move転送とした場合の計測結果である．時間の推移に従って稼働率のピークを示すプロセッサが移動していく．あるタイムスライスでは，稼働率の高い1台を除き他の3台は著しく稼働率が低い．これは全move転送戦略を用いた場合の特色であり，ジョブに依存しないことが実験的に確かめられている．図右は，最適転送属性を与えた場合の計測結果である．時間の推移に従って全プロセッサの稼働率が平均化していく．実行時間を比較すると，図左のおよそ1/3程度と短い．

これらのグラフは，multiPSI上の性能視覚化ツールparagraphによって出力された．

6 まとめ

紙面の都合によりKL1を対象とする他の動的負荷分散方式[4, 5]と比較することができなかったが，本方式の特徴は，アプリケーションからプログラムと負荷分散戦略を分離することにある．これによって，負荷分散のチューニング作業にかかる負担を軽減できる枠組が示された．

今後の課題としては，(準)最適負荷分散戦略をいかに求めていくかである．プログラムの分割パターンと各セグメント与える転送属性の組を考えると探索空間は非常に大きいことが分かる．探索空間を絞りこむ1つの方法は，セマンティックな情報を利用することである．プロセス間の通信形態を拡張されたタイプ推論で求める研究[7]が利用できると考えている．

なお，本研究は第5世代コンピュータプロジェクトの一環として行なわれた．

参考文献

[1] 神田陽治: プログラムとプロセスの再配置による並列マシン用動的負荷分散方式, 情報処理学会論文誌, Vol.31, No.12, 1990.

[2] 越村三幸, 藤田博, 長谷川隆三: KL1におけるメタプログラミング, 記号処理 57-2, 1990.

[3] 田中二郎, 的野文夫: 変数管理をするGHCの自己記述, 信学会 COMP88-5, pp.41-49, 1988.

[4] 古市昌一, 瀧和男, 市吉伸行: 疎結合並列計算機上でのOR並列問題に適した動的負荷分散方式とその評価, Proceedings of KL1 Programming Workshop '90, ICOT, pp.1-9, May 1990.

[5] 古市昌一, 中島克人, 中島浩, 市吉伸行: スタック分割動的負荷分散方式とマルチPSI上での評価, Proceedings of KL1 Programming Workshop '91, ICOT, pp.51-58, May 1991.

[6] Goto,A. et al: "Overview of the Parallel Inference Machine Architecture(PIM)", Proceedings of FGCS '88, Vol.1, pp.208-229, 1988.

[7] Shin,D.: "Towards Realistic Type Inference for Guarded Horn Clauses", Proceedings of Joint Symposium on Parallel Processing '91, pp.429-436, 1991.

[8] Taki,K.: "The parallel software research and development tool: Multi-PSI system", Programming of Future Generation Computers, North-Holland, pp.411-426, 1988.

[9] Ueda,K., Chikayama,T.: "Design of the Kernel Language for the Parallel Inference Machine", The Computer Journal, 1990.

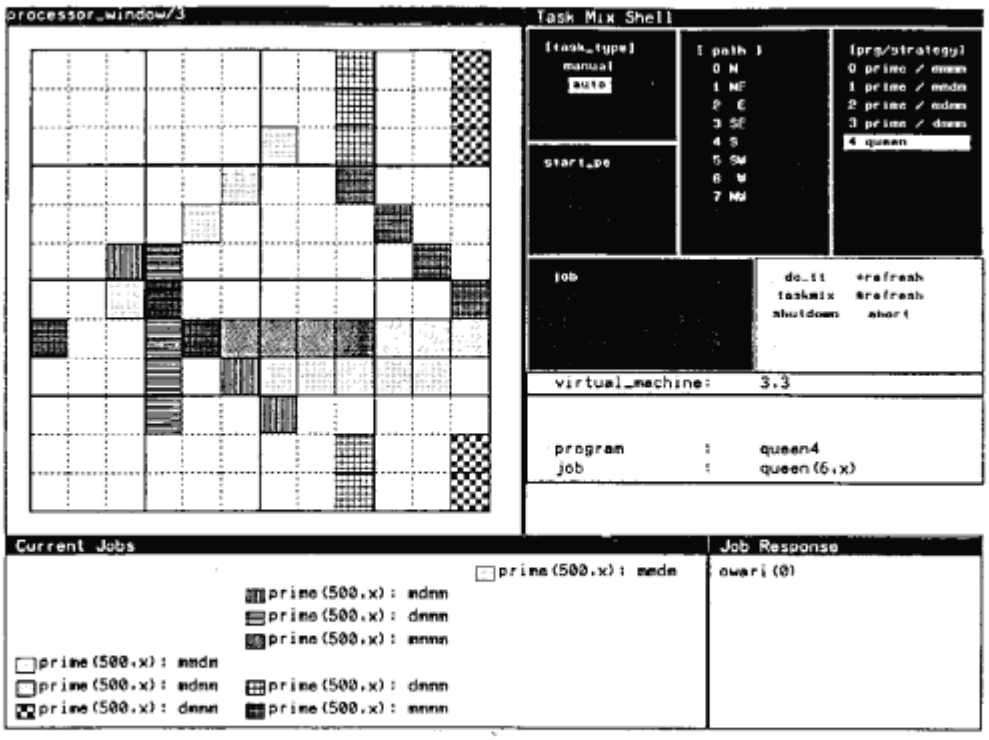

図 4: 性能管理 OS の表示画面

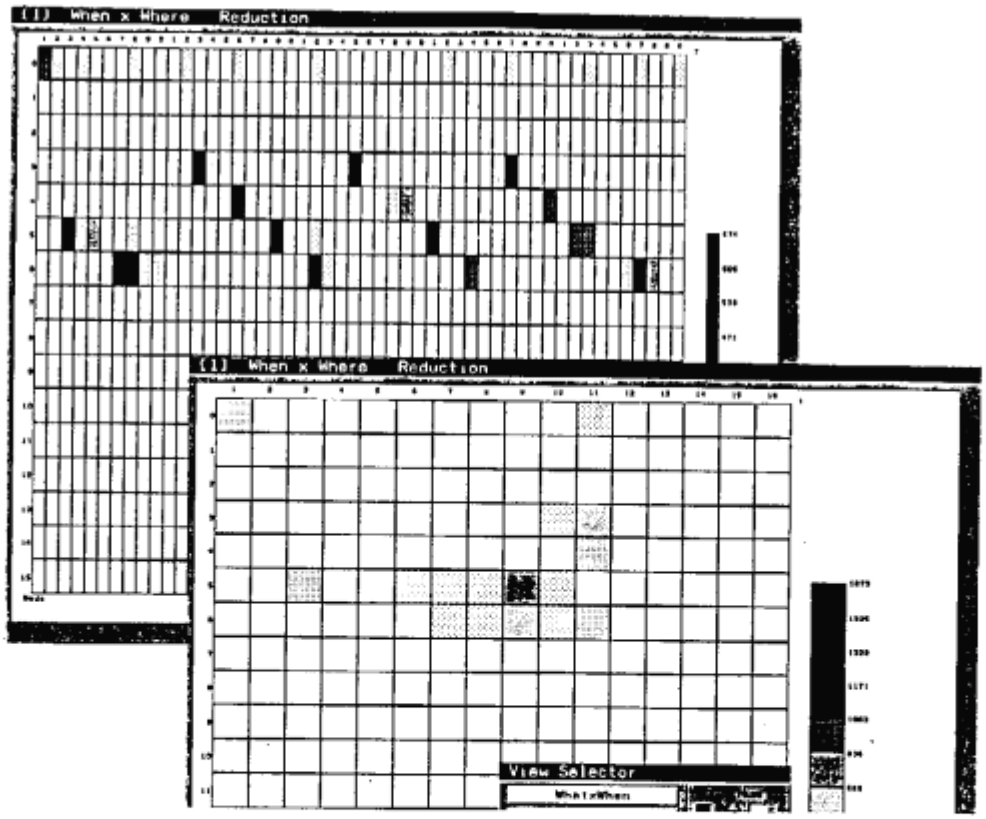

図 5: primes: 負荷分散比較