システムソフトウェアと
オペレーティング・システム 61−10
（１９９３．８．１８）

超流動 OS のための管理情報共有機構 (MetaShare) の設計

田沼 均　　平野 聡　　須崎 有康　　濱崎 陽一　　塚本 亨治

電子技術総合研究所

システムを管理するにはそのシステムの全体状況を把握していることが必要である。しかし超並列システムにおいては構成する PE が百万台規模にも及ぶのでシステム全体の状況の把握は非常に困難である。本論文では超並列システムの管理情報を能率的に収集し、適切に分配する管理情報共有機構 —MetaShare についてその基本概念と大域的仮想仮想記憶と組み合わせて使用することを想定したプロトタイプシステムについて述べる。

An Administrative Information Management System for Massively Parallel systems: Its Basic Concepts and Basic Design

TANUMA Hitoshi HIRANO Satoshi SUZAKI Kuniyasu
HAMAZAKI Youichi TSUKAMOTO Michiharu

Electrotechnical Laboratory

Operating systems must know all conditions about underlying computer systems. But it is difficult to know under massively parallel multicomputer system environment because the information is spread over millions of processing elements. We propose a system administrative information management system—Metashare. This system efficiently collects system administrative informations from each PE and effectively distributes to suitable parts. This paper describes its basic concepts and basic design.

1 はじめに

現在、百万台規模のPEの疎結合の構成をとる汎用超並列システム用オペレーティングシステム「超流動OS」を開発中である[1]。「超流動OS」の目標はネットワークトポロジへの依存性が高く規則的な動作をするデータパラレルのプログラムや、規則性が低いコントロールパラレルのプログラムといったさまざまなパラダイムのプログラムを混在して実行させる環境下で、変化する並列度や粒度などに合わせ情報(プログラム、データ等)を流動させることにより、非常に多数のPEやアクティビティを効率良く管理することである。(図1)

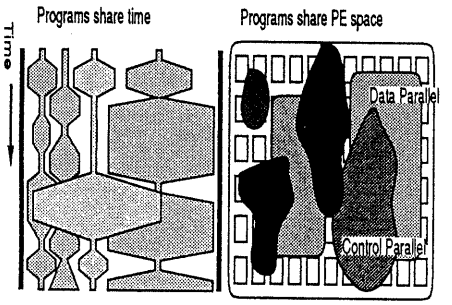

図1: 時間と空間を分割共有する超流動OS

本論文で提案するMetaShareは、超流動OSの中でシステム管理情報の収集と分配を受け持つ機構である。超並列システムにおいて管理情報は百万台規模の各PEの中に散在して存在する。またこれらを全て集めると非常に膨大な量となる。MetaShareとは、これら散在する管理情報を効率良く収集し、各サブシステムが使いやすいように抽象化など適当な処理を施し、情報を必要としているシステムに対し効率良く分配する機構である。

以下、2章でMetaShareの必要性を述べた後、3章でMetaShareの基本概念を述べ、4章で超並列システム用の仮想記憶管理システム(GVVM)を利用者とした場合のMetaShareプロトタイプの設計について述べる。

2 管理情報の大域的な管理機構の必要性

OSの目的はシステムの資源を管理しシステムを効率良く運用することにあるが、このために多くの管理情報を使用する。例えばプロセススケジューリングにおいてはプロセスの性能を損ねないためにプロセスの優先度などのプロセスの属性、CPU使用の公平性をはかるためのCPUの割り付け時間、メモリ管理のための実メモリおよび仮想メモリの割り付けマップ、各メモリ領域のアクセス権、仮想記憶のための各仮想空間の使用頻度などがある。システムを効率良く運用するにはこれら管理情報を適切に管理しシステムの状態を正確に把握していなければならない。

超並列システムを管理するOSを考えるとシステムの管理情報を適切に管理しシステムの状態を正確に把握することは通常のOSに比べてより重要な問題となり、またより困難な問題である。

例えばプロセスのスケジューリングを考える。超並列システムにおいてプロセスは複数PEにまたがり空間的広がりを持つ。プログラムによってはデータパラレルのプログラムのようにPE相互で強い同期性を有するプログラムも存在し、各PEでのスケジューリングは他のPEと協調して行なう必要がある。またPE空間にプロセスを割り当てる場合、プロセスの空間特性(トポロジなど)、システムの負荷分布、システム内での通信の状況などの管理情報を適切に使用しなければシステムの性能を損ねる結果となる。

また実記憶の制限を越えた大きな問題を解くという要求は常に存在するため超並列システムにおいても仮想記憶機構が必要となる。しかし超並列システムの各PEに現在広く使われている単一CPU用の仮想記憶機構の適用は現実的ではない。超並列システムの百万個程度ある各PEにディスクなどの二次記憶をつけるのは物理的制約から非常に困難であると予想されるため、いくつかのPEをまとめて一つの二次記憶を使うこととなる。このことは二次記憶へのバンド幅が減少するため仮想記憶の性能を著しく損ねる。またプログラムの有するメモリ使用の偏在性と局所性が原因となって小数のPEが大量のメモリを消費する現象、いわゆるメモリのファットスポットの問題が生じる。これらの問題の解決をはかるためにシステムの大域的情報を利用してシステム全体でのメモリの有効利用をはかる仮想記憶機構、例えばGVVMのようなものが必要となる[2][3]。

このように超並列システムの管理を行なうには管理情報を適切に利用していく必要があるが、通常これら管理情報はシステムの各PEに分散して存在する。超並列システムでは情報収集の対象となるPEは例えば百万台といった極めて大量に存在するため効率の良い収集方法が必要である。また実際に使用する場合、大量のPEからの情報

を生のまま提示されたのではその情報の利用のためのコストが膨大となることが予想される。このため収集した情報に何らかの抽象化を施し、それぞれ利用の目的に合うように提示される必要がある。

さらにこれらシステム管理情報は、PEの負荷状況、メモリの使用状況、ネットワークの使用状況、プロセスの展開状況、プロセスの属性などOSを構成する各サブシステム(スケジューラ、記憶管理機構等)で共通して使用される情報であるか、補助的にでも使用すれば管理の性能向上が期待できる情報である場合が多い。したがって超並列システムにおいて管理情報を一元的に収集、管理する機構を設ければ、個々のサブシステムで重複して行なっていた管理情報管理の手間が省け、さらにより広範囲の情報を利用できることとなりより効率的な管理を行なうことが可能となる。

3 MetaShareの基本概念

以下、超並列システムのための管理情報管理機構(MetaShare)の基本概念について述べる。

3.1 MetaShareの目的

MetaShareの目的は次の通りである。

1. 管理情報の効率的運用

   超並列システムを管理するには各PEに散在する管理情報を利用しなければならない。百万台規模のPEを持つシステムにおいては個々のPE上にあるOSの要素がそれぞれ独立に百万台のPEから一つ一つの情報を収集してくると膨大なコストがかかる。また収集された情報も膨大な量におよび一つ一つのPEで独立して処理を行なうのは得策ではない。

   MetaShareは管理情報の収集、分配を受け持ち、必要な場所に必要な情報を素早く提供することを第一の目的とする。収集の過程で処理を行なうことにより超並列計算機の計算パワーを利用できるので、必要な情報を望ましい形態で素早く提供することが期待できる。

2. 管理情報の共用

   収集した管理情報は複数のサブシステムで共用できるものが多い。また超並列システムからの管理情報の収集はコストがかかるので、サブシステムかそれぞれ独立に行なったのでは不経済である。さらに管理情報を統一して一元的に管理することにより、より多くの情報の利用が容易となり管理の効率化が期待できる。

3.2 MetaShare設計上の問題点

MetaShareを設計する上で以下の点を考慮する必要がある。

1. 取り扱うデータ

   MetaShareは現在、主に以下の情報を扱うことを想定している。

   - PE空間の割り当て状況

     各PEがどのプロセスに割り当てられているかを示す情報である。プロセスの割当の時に使用される。さらに同一のプロセスに属するPEはその挙動が似ているなど何らかの関係があることが期待できる。そこでこの情報をスケジューリングや仮想記憶管理などに使用すればより効果的な管理ができるものと期待できる。

   - PEの負荷状況

     各PEのCPUを負荷状態を示す。システム全体でできるだけ負荷が均衡していた方がシステムの効率が良いため、プロセスの割り当てで使用し、またプロセス内での各PEの割り当てもこの情報を利用して行なう。あるPEの負荷が高いということはそのPEに対し多くの仕事が割り付けられていることを示すので、PEの割り付け以外の管理においても負荷の高いPEに仕事が割り付かない様にすべきであるので、システム管理の全体にわたってこの情報は使用される。

   - PEの実記憶、仮想記憶の使用状況

     メモリの使用状況はCPUの負荷とともに基本的な資源の割り当て状況を示すものであり、メモリが逼迫したPEでは新たな仕事を効率良く行ない得ない。したがってシステムの各種の資源を管理する上でこの情報は基本的な情報として使用される。

   - ネットワークの使用状況

     超並列システムでは各PEが密接な通信を行なって各種処理を行なうため、輻輳などネットワークの障害は著しい性能低

下をきたす。したがって各種管理システムは通信が多く起こりネットワークに対し過大な負荷をかける可能性がないように資源の割り当てなどで配慮する必要がある。

2. データの抽象化

MetaShareが取り扱う情報は、膨大なPEにより構成される。しかし実際に情報を利用することを考えると、その様な情報が無加工で与えられると検索や処理の面で非常に大きなオーバヘッドを伴う。そこでMetaShareは利用するサブシステムに対しそのサブシステムにふさわしい精度や抽象度で情報の提供を行なう機能が必要となる。

例えばPEの負荷状況を扱う場合、収集される情報は一つ一つのPEの負荷値により構成される。しかし実際に利用される場合は一つ一つのPEの負荷値を全体にわたって必要とする場合は少なく、例えば全体を見る場合は大まかな状況がわかれば良く、また各PE個々の負荷値が問題となる場合はシステムの一部の限られた領域を対象とすることが多い。MetaShareの情報の提供の形態としては階層構造で管理し、上位階層では全体を大雑把に見るために構成要素はPEをまとめたグループを単位としててそこに含まれるPEの負荷値の平均値や最大値または最小値をそのグループの代表値として使う。上位階層にいくほど全体が見ることができるが一つ一つの値に集約されるPEの個数が多くなり、下位階層にいくほど構成要素のPE数が減って具体的な値に近付くが一度に見える範囲は狭くなる様な抽象化の方法が考えられる。

3. 情報の追随性

MetaShareの使用目的の一つとしてシステムの全体状況の把握が考えられる。この場合全体の状況を適切に把握することが目的であり、個々のPE上の細かい変化を厳密に追従する必要はない。オーバヘッドを考慮すると一定の時間間隔でデータの更新を行ない、ある程度の誤差を許容した方が得策である。

4. 情報に対するバイアスの付加

MetaShareの使用法としてプロセスのアロケーションの様にシステム全体からMetaShareを用いて使用可能な資源の探索を行なうことはよくあるが、複数の探索要求に対して同時並行的に探索を行なうと同じ資源を重複して返すこととなり割り当てが衝突し、効率を損ねる。この場合、資源がうまく分散されるように予測や変更など情報に対してバイアスを付加することを考慮する必要がある。

3.3 MetaShareの構成

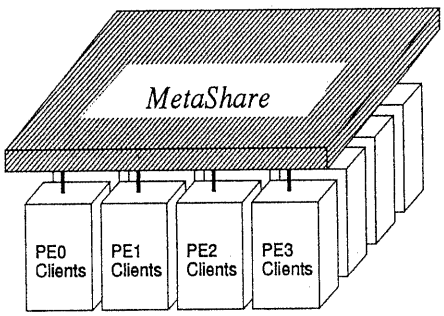

図 2: MetaShareの全体

MetaShareは概念的には管理情報を流通させるメカニズムとして振舞う。MetaShareを利用するサブシステムはMetaShareを通して管理情報を収集し、MetaShareを通して管理情報を共有する。またMetaShareは各PEから個別に管理情報を収集し、PE上のMetaShareを利用するサブシステムに対し配布する(図2)。

MetaShareは各PEに存在するMetaShare Agentの集合として構成される。各MetaShare Agentはネットワークを通じて他PEと通信し管理情報のやりとりをすると共に、そのやりとりの過程で情報を抽象化する。また自PEの管理情報を収集し、自PE内の各サブシステムに対し管理情報の提供を行なう(図3)。

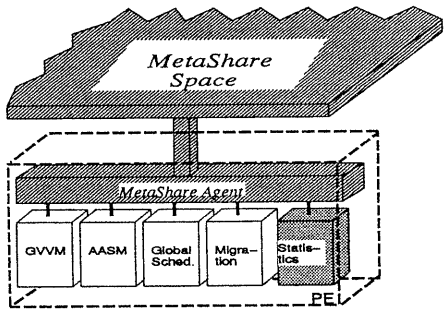

図 3: PE内での構成

MetaShareは次の4つのモジュールにより構成される。

1. Metainfo DB

   管理情報蓄積用データベース。自PEの管理情報を蓄積すると共に抽象化された管理情報や抽象化処理の中間形態の蓄積や、さらに他PEの管理情報の蓄積、問い合わせの答のデータのキャシュとして働くモジュールである。

2. Metainfo Exchanger

   他PEとの管理情報の交換を行なうモジュール。交換した情報の蓄積場所としてはMetainfo DBを使う。情報交換を行なう過程で他のPEのMetainfo Exchangerと協調して情報の抽象化処理を行なう。

3. Statistics

   自PEの管理情報の収集を行なうモジュール。収集した情報はMetainfo DBに蓄えると共にMetainfo Exchangerを通して他PEに配布する。

4. Local Server

   自PEのMetaShareを利用するサブシステムに対し、問い合わせを受け検索を実行し、情報を提供するモジュール。自PEや他PEののMetainfo DBを直接検索して必要な情報を得る。また効率の上から問い合わせの中間結果を保存しておくことが望ましい場合にはMetainfo DBに蓄える。

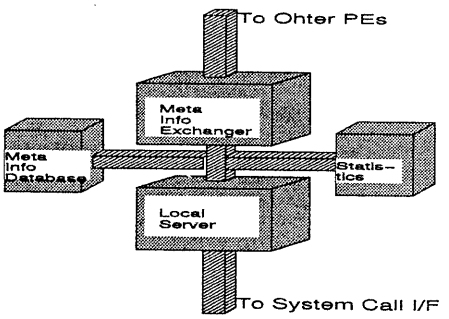

図4: MetaShare Agentの構成

4 MetaShareの実現

現在、MetaShareの利用者を大域的仮想仮想記憶としてプロトタイプを作成し、MetaShareの構成や性能を調査検討している。以下ではこのMetaShareプロトタイプについて述べる。

4.1 MetaShareへの要求

大域的仮想仮想記憶(GVVM)は特定のPEでメモリが不足し仮想記憶が動作する際、個別の各PEで仮想記憶を行なうのではなくシステム全体で使用頻度の低いページをページアウトし、さらにPEの中で使用頻度の低いメモリ領域をスワップアウト領域として使用することにより、仮想記憶の性能の大幅な向上をはかるサブシステムである。

GVVMではメモリの使用が逼迫してきた時、スワップアウトを行なうために全システム中からメモリ使用頻度の低いPEを捜し出す必要がある。この探索にMetaShareを使用する。探索の手法はさまざまなものが考えられるが、ここではGVVMに対しMetaShareがシステムのページの使用状況のマップを示し、そのマップを元にメモリ使用頻度の低いPEを探索する。

探索の基礎となる情報は各PEのページの使用頻度であり、ページャがこの情報を所有している。MetaShareは各PEに存在するページャよりPEの使用頻度の情報を受けとる。この際、必要となるのは使用頻度の低いページの多寡が問題となるので使用頻度の低い方から一定個のページアドレスとページの使用頻度情報を受けとる。

MetaShareはシステムの状態に応じて常に最新のデータを提供できることが望ましい。しかしページの頻度はページにアクセスされる毎に新しくなるので厳密にこれに追随していくには多大なコストがかかる。GVVMがページアウトのために探しているページは必ずしもシステム中の最低の使用頻度を持つものである必要はないため、情報の更新はある程度の時間間隔をもって行なう。

4.2 MetaShareの構成

4.2.1 情報の表現

サブシステムに情報を提示する際、百万個あるPEの情報をそのまま提示したのでは転送や処理のコストがかかり過ぎる。全体傾向を見る場合、全体の状態が大雑把に把握できれば良く、また細かく見る場合は限られた領域のデータが見れれば良い。そこで階層構造によって全体を表現する(図5)。

リーフに相当する部分は各PEである。PEを一定数個集めて1つの領域を構成する。さらにそ

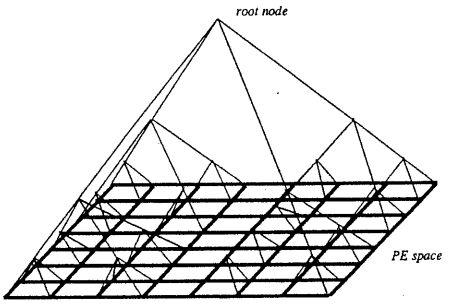

図 5: 情報の表現

の領域を一定数個集めて上位の領域を構成する。この操作を繰り返すことにより最終的に全体を表現する。ルートでは全体を見ることができるが個々の構成要素はその下位の領域を構成する領域の代表値である。

4.2.2 Statistics

ページ頻度情報の収集は各 PE の MetaShare 内の Statistics モジュールが行なう。一定時間間隔毎に自 PE のページャに対し頻度情報を要求する。それに答えてページャが使用頻度が低いページを一定個、ページアドレスと使用頻度を組にして送り返す。受けとったデータは Metainfo DB に蓄えると共に Metainfo Exchanger を通じて他 PE に配布する。

4.2.3 Metainfo DB

Metainfo DB モジュールには各木構造のノードの管理するデータが蓄積される。ノードの管理するデータとは、ノードが代表する領域に対しその領域に含まれる下位の領域を代表するノードにより構成されるマップデータである。マップの個々の要素はその領域を管理するノードのアドレス (MetaShare Agent によって識別される木構造状の位置)、その領域に含まれる PE の数およびそのノードの代表する領域のメモリ使用頻度の 3 つ組よりなる。

木構造をとると全体の状況はルートノードのある PE にのみ蓄えられることとなるので、ルートノードを持つ PE にアクセスが集中することが予想される。そこで全体のマップや広域情報を示すマップを 1 レベルまたは数レベル下位のノードに分配し、その下位のノードに対応する PE の Metainfo DB に蓄えておき、仮想的なルートノードをその PE 内に作る。これによりルートノードの分散が可能となる。

4.2.4 Metainfo Exchanger

Metainfo Exchanger には図に対応する木構造が埋め込まれ、各ノードに対応するサブモジュールより構成される。Metainfo Exchanger の動作は次の通りである。

1. 各ノードは一定時間間隔で自分の管理するメモリ使用頻度マップを自分の属する上位のノードに対し送る。
2. 下位ノードからページ頻度情報マップを受けとり、受けとったマップから自分の管理するメモリ使用頻度マップの領域に合わせて領域を再構成して領域の代表値を計算し直し、自分の管理するメモリ使用頻度マップを更新する。
3. 更新したマップを Metainfo DB に格納する。
4. ルートに近い上位のノードではルートより大域マップが配布されてくるので、それを受けとり Metainfo DB の仮想ルートノードの情報を更新する。

4.2.5 Local Server

Local Server は GVVM より要求を受けて検索を行なう。GVVM からは調べたい領域を示す木構造上のノードアドレスの指定を受ける。このノードアドレスを元にノードが自 PE 上のものであれば自 PE の Metainfo DB を検索して、また他 PE のノードであればそのノードを含む PE 上の Metainfo DB を検索する。Local Server はノードが自 PE にあるか他 PE にあるかに関わらず直接、Metainfo DB を検索する。

GVVM は使用頻度の低いページの検索に使うので、検索は通常次のように動作する。

1. GVVM はまずシステム全体の状況を知るためにノードとしてルートを指定して Local Server に対しメモリ使用頻度のマップを要求する。Local Server はルートノードもしくは仮想ルートノードが自 PE にあれば自 PE の Metainfo DB に、他 PE にあればルートノードを含む PE の Metainfo DB に問い合わせ、マップを得る。このマップの構成要素は多数の PE を含む領域においてのメモリの使用頻度である。

2. 得られたマップを元にメモリ使用頻度の低い領域を選ぶ。選んだ領域に含まれるPE数が1の場合、そのノードに対応するPEが求めるPEである。また領域に含まれるPE数が2以上の場合、選んだ領域のメモリ使用頻度マップを得るため、選んだ領域のノードアドレスを指定してLocal Serverに対しマップを要求する。

3. Local Serverは指定されたノードを含むPEのMetainfo DBからノードアドレスに対応する領域のマップを取り出す。再び2を行なう。

4.3 検討

4.3.1 代表値について

マップの構成要素は領域の状況を示す代表値をとるが、この代表値を何にするかはGVVMが何を必要とするかにより変わってくる。

例えば単純にシステム全体の中で最も使用頻度の低いページを探すのであれば代表値はその領域内のPEが有する最低使用頻度のページとそのページアドレスとすれば良い。さらに代表値を最低使用頻度のページにとればルートノードに最低使用頻度のページの候補が集まるので前述のように問い合わせの際に木をトラバースする手間は不要となる。

しかしあるPEでページアウトが起こり始めたということはそのPEでのメモリが逼迫しているということであり、再びそのPEでページアウトが起こることが予想される。また同一のプログラムのもとで動いている他のPEも似た挙動をとることが予想されるためそのPEが割り当てられた近傍の領域では他のPEにおいてもページアウトが起こる可能性が高い。

このことより検索は全システム中から最低使用頻度のページを一つだけ探してくるよりは、継続してスワップアウト領域として受け入れてもらえる領域を探す方が得策である。そこで各PEにおいて使用頻度の低い順に一定個のページを選び、その平均使用頻度をPEの代表値とし、領域の代表値は領域に含まれるPEの代表値の平均とした。このようにすれば各マップはその領域においてのページの受け入れ可能な余裕度を表現できる。

4.3.2 性能の見積り

MetaShareを使用した場合の性能について検討する。システムとしてネットワークはメッシュ結合、PE数は$M(=N\times N=2^{2n})$、PE間のネットワークの転送速度をsとする。通信の形態はストアアンドフォワード、ルーティングはY軸方向を動いた後X軸方向に移動する。MetaShareの木構造の構成は隣接4ノードをもって領域を構成する。つまり隣接する4つのPEにより1つの領域を作り、それらの隣接する4つの領域を持って新たな上位の領域を構成し、このような構成を続けてシステム全体を覆う。また管理のノードはその領域の最も右上のPEが受け持つものとする。したがってシステム中の最も右上にあるノードがルートとなる。マップの更新間隔をt、各ノードのマップの大きさはp、平均値の計算に要する時間をcとする。

まずMetaShareの全体の更新に要する最大時間を考える。木の深さはn段、同じ階層のノードは並列処理可能なのでマップ更新のための処理時間はncとなる。ルートの階層を0としてk階層の領域において最も遠い直下のノードからその領域の管理ノードまでの距離は$2\times 2^{n-k}/2=2^{n-k}$、一つのリンクにおける転送時間は$p/s$であるから総計は

$$(\sum_{k=0}^{n-1}2^{n-k})\times p/s+nc=(2^n-2)p/s+nc$$

転送量の最大は全体の更新が一時にかかった時のルートノードを持つPEが受ける情報の量である。ルートノードを持つPEはそれ以下の全ての領域においてもノードを持つPEなので各領域に含まれる下位ノードの数は4、内一つは自分であり転送の必要はないので$3p\times(n-1)=3(n-1)p$。これがtの間隔で起こるので単位時間当たりの最大転送量は$3(n-1)p/t$となる。

仮定したルーティング方法をとるとルートノードのあるPEにマップデータを転送する際、同一のY-軸上にあるPE以外全てルートノードのあるPEから右に出るリンクが使用される。したがってこのリンクが最も転送量が大きく$2(n-1)p$となり、これがt間隔で起こるので単位時間当たりの一本のリンクの最大転送量は$2(n-1)p/t$となる。

5 おわりに

本論文では超並列システムにおいて管理情報を収集、管理するシステムMetaShareの基本概念、および大域的仮想仮想記憶を利用者とした場合の

プロトタイプシステムの基本設計について述べた。

今後は本稿で述べたMetaSharedプロトタイプをシミュレータ等の上で実現し、評価を行なっていく予定である。

謝辞

本研究はリアルワールドコンピューティング計画の一環として「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。関係各位に感謝する。

参考文献

[1] 平野, 田沼, 須崎, 濱崎, 塚本. 超並列用システム用オペレーティングシステム「超流動OS」の構想. 情報処理学会研究会報告 93-OS-58,Vol. 93, No.27, 17-24, 1993.

[2] 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, 237-244,1993.

[3] 平野, 田沼, 須崎. 超流動OSの大域的仮想仮想記憶におけるページ探索方の比較. SWoPP'93 OS 発表予定,1993.