超並列環境向きトラバースアルゴリズムを用いたプロセス移送

大澤範高[†,☆] 弓場敏嗣[†]

要素プロセッサ（ノード）数が100万規模の超並列計算機を利用する超並列環境に向いたプロセス移送方式を提案し，評価する．高いノード次数を持つネットワークを構成することとノード間リンクの通信容量を非常に大きくすることはハードウェアコストから難しい．そこで，超並列環境においては，通信容量の不足による遅延を防ぐために，プロセス移送を行うことによって通信の局所性を高め，利用する通信容量（通信コスト）を小さくすることが重要である．プロセス間通信ネットワークをたどりながら移送先を順に決定するという移送アルゴリズムを提案し，それに従った移送による通信コストの改善効果を解析する．さらに，改善効果をシミュレーションによって確認し，結果を考察する．また，超並列環境におけるプロセス移送と耐故障性および楽観的実行との関連を述べ，プロセス移送のためのオーバヘッドの問題が超並列環境においては相対的に小さくなることを論ずる．

Process Migration Using Traverse Algorithm in a Massively Parallel Environment

NORITAKA OSAWA[†,☆] and TOSHITSUGU YUBA[†]

A process migration algorithm in a massively parallel environment, which uses a massively parallel computer with more than one million element processors, is proposed and evaluated. It is difficult from a viewpoint of hardware cost to construct a network whose nodes have large degrees and to significantly increase communication capacity between nodes in a massively parallel computer. However, a network topology with a fixed degree is not scalable with respect to communication capacity. Insufficient communication capacity causes communication delays. In order to prevent communication delays, it is important to utilize locality. Locality is improved by process migration. This paper proposes a process migration algorithm which traverses the inter-process communication network and determines the destination of a migration in order. Next the reduction of communication costs is analyzed. The result of the analysis is confirmed by simulation and the simulation studies are examined. This paper also describes the relationship between process migration and other facilities, such as fault tolerance and optimistic execution, and discusses the effective overhead of process migration is low in a massively parallel environment.

1. まえがき

近い将来，要素プロセッサ（PE）数が100万規模の超並列計算機の実現が期待される．また，ワークステーションやパーソナルコンピュータから構成されるメタ・コンピュータがメタ・コンピューティング[1]のために利用されようとしている．このような超並列計算機上で動作するオペレーティングシステム（OS）をここでは超並列オペレーティングシステム（超並列OS）と呼ぶ．超並列OSは，超並列計算機の性能を有効に利用できるようにすることが重要である．また，超並列計算機および超並列OS等のシステムソフトウェアをまとめて超並列環境と呼ぶ．

超並列計算機のすべてのPEを利用する場合に，複数のプロセスから構成されるジョブの実行が最速になるとは限らない．このため，PE数が100万規模の超並列計算機は複数のジョブによって共有されることになり，システム内に複数のジョブが同時に存在することになる．

超並列環境では，固定的な数のPEを占有するのではなく，必要や環境に応じてPEの割当ておよび解放を行うジョブが増えると考えられる．このような性質を持つ応用としては，現在でもプロセッサファームモデルに基づいた応用プログラムが考えられる[2),11)]．ま

† 電気通信大学大学院情報システム学研究科
Graduate School of Information Systems, The University of Electro-Communications
☆ 現在，メディア教育開発センター
Presently with National Institute of Multimedia Education

た，動的に実体が生成・消滅する並列離散事象シミュレーションも考えられる．さらに，メタ・コンピューティングによって解こうとしている問題にも上記性質がある．本論文では，利用されるPEがジョブの起動時に固定されるプロセッサ分割方式ではなく，ジョブの処理進行に応じてPE割当てが変更される（プロセス数が変化する）一般的な場合を議論する．

局所性を活かすためにPEの連続割当てを行い，不要時にPEの解放を行うと，使用されないPEが多数発生することが予想される．ジョブ処理を繰り返すことにともなって平衡状態に近付くとし，各ジョブが要求するPE数が同じになることはほとんどなく，ジョブの実行時間はそれぞれ独立で，割当てがPE番号の区間で行われる場合には，Knuthの50%則[6]によって，空き区間の平均数は，利用されている区間の半分になることが分かる．超並列計算機においては多数のPEが利用されないことになる．相互結合網のトポロジに応じた割当ての方法をとることなどによって有効利用されないPE数を少なくすることは可能であるが，より根本的に改善するためには，ジョブ実行中の移送が必要である．

また，ジョブを構成するプロセスのすべてをつねに実行可能状態にすることはできず，待ち状態にあるプロセスもある．このような状況でPEを有効利用するためには，多重プログラミング機能によるプロセス多重化が必要である．プロセス多重化を行う場合には，PE間の負荷の均衡を図ることが，システムの性能向上のために重要である．負荷分散を適切に行うためにも移送が必要である．

次に，通信総容量と移送の必要性の関係を考える．n 次元トーラス構造においてノード総数を M とすると1辺の長さ L は，$L = \sqrt[n]{M}$ であり，n 次元トーラス構造の直径 D は $D = n\frac{L}{2}$ となる．また，ノード間の平均通信距離 d は，トーラス構造の対称性から n が奇数のとき $d = (D+1)/2$ であり，偶数のとき $d = \frac{D}{2}\frac{M}{M-1}$ である．以下では奇数のときのみを扱うが，偶数の場合も同様に議論できる．

通信路の一方向の通信容量を v とすると，通信総容量は，

$$V = vnM/2 \tag{1}$$

である．ここで，通信路は双方向とする．

すべてのノードにプロセスが存在し，それらのプロセスが少なくとも1つの他のプロセスと通信するとすると，少なくとも $M/2$ の通信リンクがあることになる．通信するプロセスがランダムに配置されているとし，プロセス間通信の平均通信量を a とすると，通信総コストは，

$$A = adM/2 \tag{2}$$

である．ここで，ある通信の通信コストは通信距離と通信量の積と定義する．これから，

$$\frac{A}{V} = \frac{adM}{vnM} = \frac{a}{2v}\left(\frac{\sqrt[n]{M}}{2} + \frac{1}{n}\right) \tag{3}$$

である．a，v は正の値であるので，n を一定とすれば，M の増加につれて通信総コストが総容量を超えてしまう可能性が高まる．つまり，超並列計算機において，通信コストが通信容量を超えないようにするためには，n，v のいずれかを増加させるか，通信コストを減らすためにプロセス間通信距離を減らすことが必要である．プロセス間通信距離の軽減は，移送によって通信の局所性を高めることによって行われる．

ノード次数の大きなネットワークを構成することとノード間リンクの通信容量 v を大きくすることにはハードウェアコストがかかり，ノード数の多い超並列計算機においては困難がある．ワームホール（wormhole）やバーチャルカットスルー（virtual cut-through）[4]によってノード間距離による遅延は問題にならなくなってきたが，超並列計算機の効率的な利用のためには，通信コストを減らすことが必要である．

本論文では，後述のようにデータ移動を移送の有無にかかわらず行わなければならず，システム全体の情報を収集するコストが大きな超並列環境に適した，移送アルゴリズムであるトラバースアルゴリズムを提案する．このアルゴリズムによってスループット性能がいかに向上するかを評価する．

従来の分散システムのプロセス移送研究における移送の評価基準には，応答時間（ターンアラウンド時間）が広く利用されている[10),13)]．超並列計算機の利用は，処理時間短縮を目的とするものであり，移送による応答時間の短縮は必要である．しかし，超並列計算機でも共用を許すならばスループットの向上を図ることも必要である．

2. 移送の評価要因

ある期間における移送のコストとメリットの関係について検討する．移送のコストは，プロセス移送に必要な平均コスト α と期間あたりの移送すべきプロセス数 N_m の積で表すことができる．一方，超並列環境において，移送によって生まれるメリットは2種類に大別できる．局所性を高め，通信距離を減らすことによって通信コストを低減し，通信遅延を小さくできることと，負荷分散によって実効処理能力を高められることである．これは，プロセスあたりの通信コスト

の低減による処理の高速化 β と負荷分散による平均処理能力向上 γ として表すことができる．すなわち，移送によって性能が向上するためには，次の関係が成り立つ必要がある．N_c を移送のメリットを受けるプロセス数とする．

$$\alpha N_m \leq \beta N_c + \gamma N_c \qquad (4)$$

左辺は移送のコストであり，右辺は移送によって得られるメリットである．左辺と右辺の差が最大になるようにすることが重要である．

超並列環境においては，α がデータ移送のコストを必ずしも含まないことは後述する．β は，移送後の配置をどのようにするかによって影響を受ける．γ は，完全に負荷分散できた場合にどの程度近付くことができるかが問題である．本論文では，通信コストの低減の解析を行う．

3. 通信コストの低減

ジョブを構成するプロセスの地理的局所性を高めることによる平均通信コストの低減を検討する．

3.1 移送アルゴリズム

まず，例を使って移送アルゴリズムを説明する．本論文で提案するアルゴリズムをトラバースアルゴリズムと呼ぶ．**図 1** のような 2 分木の相互結合網トポロジを持った並列計算機を考える．この並列計算機上でプロセス A，B，C，D が動作しており，それらのプロセスは図のように PE に割り当てられているとする．PE 間の実線で示された線分の本数が，距離を表すとする．また，破線で結ばれたプロセス（PE）間で通信が行われており，プロセス間の通信量はすべて 1 とする．この場合に，システム全体の通信総コストは，16 である．

ここでは，A から始めて，プロセス間通信グラフ（ネットワーク）をたどってその順に移送を行う．複数のコネクション（リンク）がある場合には，それまでの通信コスト等の履歴情報を考慮する．例では，最も通信コストの大きなコネクションからたどる．このアルゴリズムが適用されると**図 2** のように配置される．通信総コストは減少し，8 となる．このようにして，移送によってシステム全体の性能の向上を図ることが可能である．このアルゴリズムでは，通信コストがつねに小さくなることは保証されない．

移送すべきプロセス集合の決定は，スナップショットガーベジコレクション[12]での生きているオブジェクトの印付けとほぼ同様である．しかし，移送の順序の決定において履歴情報を利用する点が異なっている．

例では，移送先の決定（移送起動処理）とデータ移動が同時に行われるとして説明したが，実際には，移送起動処理とデータ移動は分離して処理してよい．ここで規定するのは移送起動処理についてである．

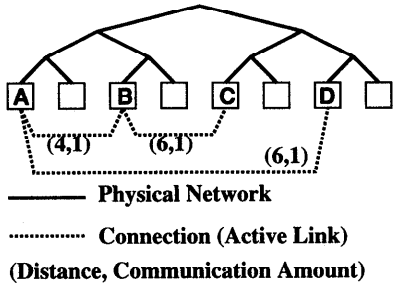


図 1 移送前の状態
Fig. 1 States before migration.

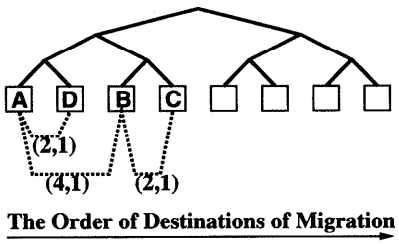


図 2 移送後の状態
Fig. 2 States after migration.

各プロセス間通信は，コネクションベースで行われるとする．コネクションレスの場合には，通信のたびに，コネクションの設定，通信，コネクションの切断が行われるとすれば，このアルゴリズムを適用できる．

システムには，移送管理サーバ（複数プロセスで構成されてもよい）が存在し，ある時点でのスナップショットにおけるすべてのプロセスは，移送管理サーバからコネクションをたどることによって到達可能とする．移送管理サーバは，プロセス群の生成・消滅を管理しており，活動中のプロセス群のルート集合はすべて移送管理サーバ（群）によって分かっている．移送処理は，移送管理サーバが管理するルート集合からプロセスを順次たどることになる．故障が存在するシステムでは，移送管理サーバに管理されていない孤児（orphan）状態になるプロセスが存在しうるが，孤児プロセスは移送する必要はなく，削除が必要なだけであり，また，削除問題は移送とは別問題であるので本論文では扱わない．

移送開始タイミングは，移送管理サーバが判断する．この判断は，タイマや負荷状況に基づいて行われるが，移送開始タイミングは，ここで扱うアルゴリズムとは分離する．ここで規定するのは，移送先の決定手順で

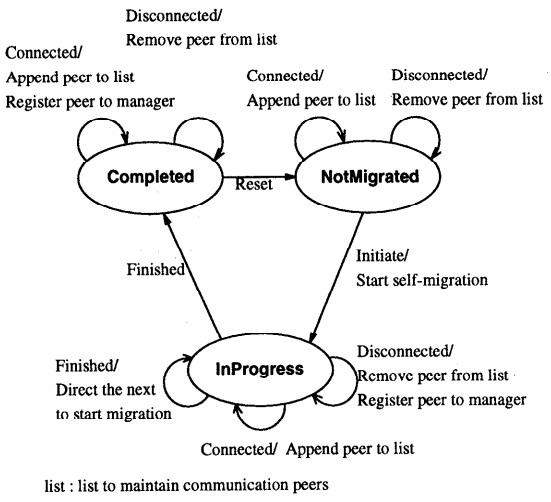


図3 状態遷移図
Fig. 3 State transition diagram.

ある．データを実際に移送する機構とシステム全体の移送完了検出部分は含まない．また，移送先の利用順序は，トポロジおよび利用形態に応じて最適になるようにあらかじめ決める必要がある．利用順序の優劣については，後の章で検討する．

プロセスの移送処理の状態遷移図を図3に示す．また，動作を C++ 言語風に記述したものを付録に示す．各プロセスの移送関連状態は，非移送状態（NotMigrated），移送起動処理中（InProgress），移送起動処理完了（Completed）に分けられる．初期状態は非移送状態である．各プロセスには，接続（Connected），切断（Disconnected），移送起動指示（Initiate），移送起動終了（Finished），リセット（Reset）という事象が発生する．接続，切断は，それぞれ，プロセス間のコネクションの接続，切断時に発生する事象である．移送起動指示は，プロセスの移送起動処理を指示するメッセージによって発生する．また，移送起動終了は，プロセスの移送起動処理が終了した際に，移送起動指示を行ったプロセスに発生する事象であり，移送そのものの完了を意味しない．リセットは，データの移動を含めて移送が完了した場合にシステムから与えられる事象である．

移送アルゴリズムは次のとおりである．非移送状態にあるプロセスが，移送起動指示を受け取ると移送起動処理を始める．そのプロセスは移送起動処理中になる．プロセスが管理している通信相手のリストと履歴を基に，移送起動指示を通信相手に順に送る．移送起動処理中に，接続，切断が発生した場合には，リストを更新する．また，切断が発生した場合には，それまでの通信相手が移送対象から外されることがないように移送管理サーバへの登録が行われる．これによって，すべてのプロセスに対して移送処理が行われる．リストのすべての通信相手の処理を完了した際には，移送起動指示を送ってきた親に相当するプロセスに移送起動終了を送る．

3.2 移送のオーバヘッド

ここでは，移送のオーバヘッドについて考察する．まず，移送に要するアルゴリズムの計算処理量について考える．本論文の提案する移送アルゴリズムでは，通信を行っているプロセスのグラフをたどり，あらかじめ決められた順序に従って移送先が決定される．プロセス数を N とすると，移送先の決定に必要な計算量は $O(N)$ となる．

次に，本方式では，移送先を積極的に通知する必要はない．移送元ノードに移送先を記録しておくことにより，必要に応じてプロセスの移送先情報を更新することが可能である．このアルゴリズムは，分散共有記憶の分散マネージャ方式によるページの移送先管理[7]と同様である．たとえば，あるノード A 上のプロセス P から，別のノード B 上にあったがノード C に移送されたプロセス Q へのコネクション設定時に B に記録されている移送先情報が利用される．いったん移送先をノード A が知った後は，A にあるプロセスから Q へのコネクション確立時には B が参照されることはない．

3.3 アルゴリズムの利点，欠点

ここで示したアルゴリズムの利点は，次のとおりである．(1) プロセス間通信グラフをたどりながら移送起動を行うことによって，個々の PE に処理を分散させることができる．(2) あらかじめ移送先の利用順序を決めておくので，負荷等に応じて最適な配置を移送処理時に計算する場合に比べて，実行時オーバヘッドが少ない．(3) プロセスの状態を増やし，バージョンもしくは「色」をつければ，移送処理を多重化させて進めることができる．(4) 移送先の順序が決まっているので，同じプロセスの移送が繰り返される場合に発生する周期の短い振動現象が起きない．

一方，欠点は以下のとおりである．(1) 移送起動指示を受け取ってから移送起動終了を送信するまでに，プロセスに故障が起きると処理が停止する．しかし，各プロセスが移送先を移送管理プロセスと通信して決定するようにすれば，移送管理プロセスがタイムアウトによって故障を検出することが可能である．(2) あらかじめ移送先の利用順序を決めておくので，最適の

配置にはならない．しかし，移送中に状態が変化する環境においては，ある時点の最適配置を利用したとしても，移送終了時において最適になっている保証は存在しない．変化する環境における実質的な効果の評価が重要である．

4. 移送の効果

移送による通信コストの低減効果を解析する．まず，プロセスの全体数を N とし，その間の ${}_N\mathrm{C}_2$ 個のリンクを考える．プロセス i からプロセス j へのリンクの距離，通信量をそれぞれ d_{ij}，m_{ij} とする．ここで，直接接続されたノード間の距離を 1 とする．

システム全体の通信総コストを

$$C = \sum_{i=1}^{N}\sum_{j=1}^{N} d_{ij} m_{ij} \tag{5}$$

と表す．

移送を行わない場合の通信総コストを C_{nomig} とし，移送を行った場合の通信総コストを C_{mig} とする．通信総コスト改善率 B を

$$B = \frac{C_{nomig} - C_{mig}}{C_{nomig}} \tag{6}$$

と定義する．

4.1 条　　件

通信量の変化はマルコフ過程とし，すべてのプロセス間通信の統計的な性質は同じとする．すべてのプロセス間での通信量を計算することによって，複数のジョブとジョブ内プロセス並列度変化およびプロセス間通信量変化を近似する．

最も基本的な場合として，$d_{ij} = d_{ji}$，$m_{ij} = m_{ji}$ で，通信量 m_{ij} が 0 または 単位量 1 のいずれかである場合を検討する．単位時間あたりの通信量の種類を増やし，通信量を考慮した場合には，2 種類の場合よりも良い改善率が得られると考えられる．通信量 0 から 0 への推移確率 p_{00} と通信量 1 から 1 への推移確率 p_{11} が解析のパラメータとなる．

4.2 活動率と平均通信相手数

通信が行われているリンクを活動リンク（コネクション）と呼ぶ．定常状態において活動リンクのリンク全体に占める割合（活動リンク率）は，$\overline{r_{a_l}} = (1-p_{00})/(2-p_{00}-p_{11})$ であり，プロセスあたりの平均活動リンク数は，$\overline{n_{a_l}} = (N-1)\,\overline{r_{a_l}}$ となる．

次に，定常状態においてプロセスが通信する相手の平均数である平均通信相手数 $\overline{n_p}$ を求める．少なくとも 1 つのリンクで通信が行われているプロセスを活動プロセスと呼び，定常状態において，活動プロセスのプロセス全体に占める割合を活動プロセス率 $\overline{r_{a_p}} = 1-(1-\overline{r_{a_l}})^{N-1}$ とする．また，定常状態における活動プロセスの平均数を平均活動プロセス数 $\overline{n_{a_p}} = N\,\overline{r_{a_p}}$ とする．この場合，$\overline{n_p} = \overline{n_{a_l}}/\overline{r_{a_p}}$ となる．

4.3 近傍平均距離と改善率

あるトポロジにおいて，移送先として利用する順序に番号をノードにふる．番号の小さい方から n 個のノードにおいて，近い番号の m ノード間の平均距離を近傍平均距離 $L(n,m)$ とする．

$$L(n,m) = \frac{1}{s(n,m)} \sum_{i=1}^{n} \sum_{j=i+1}^{min(i+m-1,n)} d_{ij} \tag{7}$$

ここで，$s(n,m)$ は，n ノード中の近傍 m ノード間のリンクの数である．

通信総コスト改善率 B は，全ノード間の平均距離 $\overline{d} = L(N,N)$ に対する実効的な平均距離の改善率によって表せる．実効的な平均距離は，近傍平均距離を基に表せると考える．その場合に，近傍平均距離のパラメータ n は $\overline{n_{a_p}}$ になり，m は，$\overline{n_{a_l}}$ 以下と考えられる．ただし，通信をしているプロセスは移送によって近傍に集まるが，推移確率に応じて通信相手が変化するので，移送後のプロセス間の実効的な近傍平均距離は，

$$l(n,m) = p_{11}\,L(n,m) + (1-p_{11})\,\overline{d} \tag{8}$$

となる．

そこで，平均距離実効改善率 $E(n,m)$ を

$$E(n,m) = \frac{\overline{d} - l(n,m)}{\overline{d}} \tag{9}$$

とすると，実際の改善率は，以下の関係にあると考えられる．

$$B \le E(\overline{n_{a_p}}, \overline{n_{a_l}}) \tag{10}$$

5. シミュレーション

前記のアルゴリズムによるシステム全体の通信コストの低減の解析結果をシミュレーションによって確かめた．通信量の推移確率パラメータを変化させてシミュレーションを行った．複数の活動リンクがある場合には，通信コストの大きなものから順に移送処理を行った．

シミュレーションの期間は，1000 単位時間とした．初期値の影響を排除するために 100 単位時間実行後にデータ採取を開始した．1 単位時間には，前記のアルゴリズムによる一連の移送処理がシステム全体に対して行われる．本論文の通信量の推移確率行列は，正則なチェーンの推移確率行列となっている．繰返し回数を i とす

ると初期値の影響は $(p_{00}+p_{11}-1)^i$ に従って減少する．図4のシミュレーションでは，$(p_{00}+p_{11}-1) < 0.95$ であり，100回の繰返しで，初期値の影響は 5.92×10^{-3} 以下となっている．

5.1 シミュレーション結果

シミュレーションの対象としたトポロジは，リング，2分木，ハイパーキューブであり，番号 i と 番号 $j(\neq i)$ のノード間距離が，それぞれ，

$$d_{ij}^{ring} = \mathrm{Min}(|i-j|, |j+N-i|, |i+N-j|)$$
$$d_{ij}^{tree} = \lfloor \log_2 (i \oplus j) \rfloor + 1$$
$$d_{ij}^{hypercube} = \mathrm{Hamming}(i, j)$$

となるように0から $N-1$ までのノード番号を付与した．ノード番号の小さい方から移送先として順に利用する．ここで，N は要素数であり，Min，$|x|$，$\lfloor x \rfloor$，$\oplus$，Hamming は，それぞれ，最小値，絶対値，x を超えない整数，ビット単位での排他的論理和，ハミング距離を求める関数である．

リング，2分木，ハイパーキューブに対するシミュレーション結果を**図4**に示す．また，図4には，PEが128台の場合の前章の解析による平均距離実効改善率の上限も示す．

リング，2分木，ハイパーキューブにおいてプロットした点におけるBの値（%が単位）の信頼度95%の信頼区間の大きさの最大値はそれぞれ1.30，0.77，0.84であった．**図5**に，信頼区間をエラーバーとして表示した．

さらに，平均通信相手数と改善率の関係を明確に示すために，図4の格子点を平均通信相手数と改善率との関係でプロットしなおしたものを**図6**に示す．

前章の解析および本章のシミュレーションから導けることを以下にまとめる．

- リング構造が最も効果が高く，2分木，ハイパーキューブの順に効果が少なくなる．
- 解析的に考えた改善率の範囲にシミュレーション結果が入っている（図4）．前章の解析は，要素数に依存しないので，超並列においても同様の議論

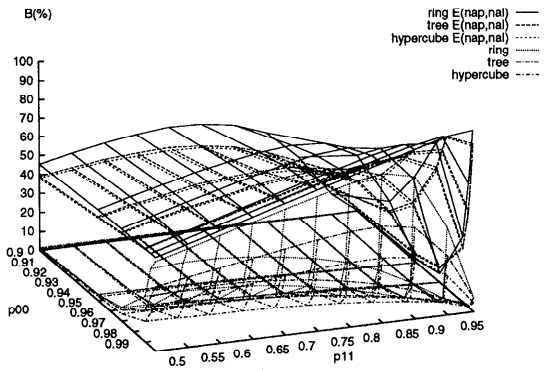


図4 128台の場合の改善率（B）と平均距離実効改善率—**ring**，**tree**，**hypercube** は，それぞれ，リング，2分木，ハイパーキューブを意味する．**E(nap,nal)** は，$E(\overline{n_{a_p}}, \overline{n_{a_l}})$ を意味する．

Fig. 4 Improvement ratio (B) and effective reduction ratio of average distance between processes in 128-PE computer – **ring**, **tree**, **hypercube** represent ring structure, binary tree structure, hypercube structure respectively. **E(nap,nal)** represents $E(\overline{n_{a_p}}, \overline{n_{a_l}})$.

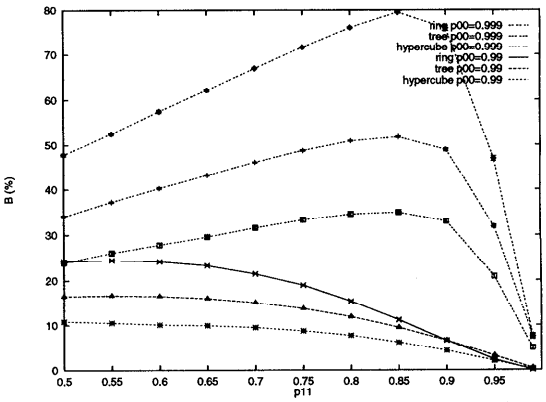


図5 128台の場合の改善率（B）の信頼度95%の信頼区間–信頼区間をエラーバーとして表示してある．

Fig. 5 Confidence interval of improvement ratio (B) in 128-PE computer.

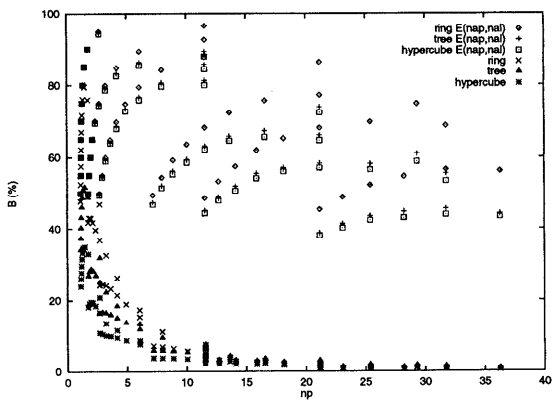


図6 128台の場合の平均通信相手数と改善率の関係—図4の格子点を平均通信相手数（np）と改善率（B）との関係でプロットしなおしたもの

Fig. 6 Relationship between average number of peers and improvement ratio in 128-PE computer – Points in Fig. 4 are plotted.

が成り立つ．

- 平均通信相手数が少ない場合には $E(\overline{n_{a_p}}, \overline{n_{a_l}})$ に近いが，増えるに従って値が小さくなる（図6）．これは，通信相手数が増えるに従って通信相手を連続に配置することができなくなり，近傍平均距離における近傍の範囲が上限に近付くためと考えられる．
- 近傍平均距離に基づいた平均距離実効改善率によって，移送先PEの利用順序を評価できる．
- プロセス間通信のグラフが木構造を構成する場合の平均通信相手数は2に近い．その場合には，本論文の簡単なアルゴリズムでも10%～40%程度の改善が得られる．平均通信相手数が大きい場合にはほとんど改善が見られない．一般に，ノード次数の大きなスイッチから構成される相互結合網を性能価格比良く構築することは難しい．したがって，合理的な性能価格比を有する超並列計算機においては，移送方式によらず，平均通信相手数が大きい場合の通信コストの低減が難しい．

6. 議　論

超並列環境を実現した場合の移送コストの問題を考えるために，耐故障性と楽観的実行について論ずる．

6.1 耐故障性

超並列環境においては，耐故障機能が不可欠である．PEあたりの平均故障間隔（MTBF）を 10^6 時間☆とすると[3]，100万PEの超並列計算機においては，MTBFは1時間となる．このMTBFよりも時間がかかる計算は，完了の見込みが少なくなる．超並列計算機は，大規模な計算の処理を目的としており，ジョブの計算時間が前記のMTBFよりも十分に小さいと仮定することは非現実的である．また，既存のワークステーションをPEとして利用するメタコンピュータの場合には，保守などによりPEの停止（以下では故障と同一視する）が少なくとも年に1回以上あると考えられるので，PEあたりのMTFBは約 10^4 時間となり，状況はより悪化する．したがって，超並列環境においては，故障の発生を前提とし，耐故障機能を備えることが不可欠となる．

耐故障機能を実現するためには，チェックポイントをとり，故障発生時に回復を行えるようにする必要がある．このチェックポイントのためにデータ複製を行うことは，移送の別の側面と考えることができる．たとえば，回復可能分散共有記憶システムにおいて耐故障性のためのデータ複製と，分散共有記憶のためのページの移動（移送）を統合しているシステム[5),9)]と同様の手法を用いることによって，移送のオーバヘッドを小さくすることができる．つまり，耐故障性のためにデータ複製を行うシステムにおいては，データ移動のための通信は，必ずしも移送のオーバヘッドとして考えなくてもよい．

超並列計算機では，すべてのPEが2次記憶装置を持つことは難しい[15)]．したがって，PEの1次記憶領域にチェックポイントデータ（複製データ）が保持されると考える[5)]．また，すべてのチェックポイントを最初から2次記憶に保存する必要はなく，性能向上のために複数の版のチェックポイントデータが異なった記憶階層に保持されるようになると考える．より古い版が安定記憶に保持されているのであれば，最新のチェックポイントは他のPEの揮発性メモリに保持されていてもよい．それが必要に応じて，より安定な記憶に移される．

6.2 楽観的実行

並列性をより引き出して利用するために，ある範囲の応用においては，緩い同期をとりながら実行を進め，不整合が発見された場合に回復を行う楽観的実行が超並列環境において重要である．楽観的実行支援は，超並列OSが備えるべき機能の1つであると考える．楽観的実行を支援する場合には，状態の回復機能が必要である．回復のためには，耐故障性の場合と同様に状態保存が必要であり，そのためにデータ移動（複製）が必要である．楽観的実行支援機能を持ったOSにおいても，移送のデータ移動は，必ずしも移送のオーバヘッドとして考えなくてもよい．

7. 関連研究

プロセス移送による動的負荷分散はこれまでにも研究されており，プロセス移送方式は，(1) だれが移送判定の主体になるか，(2) いつ移送判定を行うか，(3) どのプロセスを移送するか，(4) どこへ移送するか，という決定から分類される[14)]．

提案の方式では，移送管理サーバからすべての生きているプロセスを，あらかじめ決められた順序のノードに移送する．上記の分類では (3) と (4) に相当する部分に特徴がある．

また，移送判定には情報が必要であり，情報収集の方式は局所方式と大域方式に大別できる．局所方式は，状態情報の交換を行わない方式であり，本論文の提案

☆ ほとんどのハードウェアモジュールのMTBFは，1985年に1年程度であったが1990年には10倍程度になったと文献3)に記述がある．そこで，さらなる改善を見込んで，100年≈10^6時間とした．

が利用する方式である．大域方式は，システム全体の情報を収集し，それに基づいてプロセス移送を行う．超並列環境においてはシステム全体の情報を収集するオーバヘッドと遅延が従来よりも大きい．また，正確な全体の状況を把握するためには情報収集の間隔を短くしなければならず，オーバヘッドが問題になる．局所方式によって大域的な情報収集のオーバヘッドを減少させることができる．

8. あとがき

まず，PE数が100万規模の超並列計算機においては高いノード次数を持つネットワークを構成することとPE間リンクの通信容量を非常に大きくすることはハードウェアコストから難しく，通信容量の不足による遅延を防ぐためにプロセス移送を行うことが重要であることを示した．

次に，超並列環境に適した，プロセス間通信グラフをたどりながら移送先を順に決定するという移送アルゴリズムを提案した．提案アルゴリズムによる通信総コスト改善効果が超並列計算機のネットワークトポロジと要素プロセッサの利用順序によってどのような影響を受けるかを解析し，シミュレーションによって評価した．

提案の方式は，プロセスを動的に生成・削除する複数のアプリケーションが同時に実行され，耐故障性・楽観的実行支援のために状態保存を行う必要がある超並列環境において特に有効である．また，他のプロセスの負荷状況なしで移送を行えるので，システム全体の情報収集のコストが高い超並列環境に適している．

本論文では通信コストを中心に解析および評価をしたが，負荷分散による性能向上についても検討を行う必要がある．また，超並列環境におけるプロセスの寿命も言語処理系などにおけるオブジェクトの寿命[8]と同様の性質を持つと考えられる．これらの性質を利用した世代別管理により性能向上を図ることが可能であり，その効果の評価が必要である．

謝辞 移送の必要性について有益な示唆をいただいた電気通信大学大学院情報システム学研究科佐藤直人氏に感謝いたします．

参考文献

1) Catlett, C. and Smarr, L.: Metacomputing, *Comm. ACM*, Vol.35, No.6, pp.45–52 (1992).
2) Dancea, I.: Development and evaluation of processor farm algorithms/software for robot sensing applications, *IEEE Instrumentation and Measurement Tech.*, pp.416–420 (1995).
3) Gray, J.: Census of Tandem System Availability Between 1985 and 1990, *IEEE Trans. Reliability*, Vol.39, No.4, pp.409–418 (1990).
4) Hwang, K.: *Advanced Computer Architecture: parallelism, scalability, programmability*, McGraw-Hill (1993).
5) Kermarrec, A.-M., Cabillic, G., Morin, A.G.C. and Puaut, I.: A Recoverable Distributed Shared Memory Integrating Coherence and Recoverability, *Proc. IEEE 25th Int'l Symp. on Fault-Tolerant Computing*, pp.289–298 (1995).
6) Knuth, D.E.: Fundamental Algorithms, *The Art of Computer Programming*, Vol.1, Addison-Wesley (1973).
7) Li, K. and Dudak, P.: Memory Coherence in Shared Virtual Memory Systems, *ACM Trans. Computer Systems*, Vol.7, No.4, pp.321–359 (1989).
8) Lieberman, H. and Hewitt, C.: A Real-Time Garbage Collector Based on the Lifetimes of Objects, *Comm. ACM*, Vol.26, No.6, pp.419–429 (1983).
9) Osawa, N. and Yuba, T.: Lazy and Differential Replication in a Recoverable Distributed Shared Memory System, *High-Performance Computing and Networking*, LNCS, Vol.1401, pp.698–707, Springer-Verlag (1998).
10) Shivaratri, N.G., Krueger, P. and Singhal, M.: Load Distributing for Locally Distributed Systems, *IEEE Computer*, Vol.25, No.12, pp.33–44 (1992).
11) Tambouris, E., van Santen Peter, P. and Marsh, J.F.: Performance evaluation of a processor farm model for power system state-estimators on distributed-memory computers, *Computing Systems in Engineering*, Vol.5, No.4–6, pp.479–487 (1994).
12) Yuasa, T.: Real-time garbage collection on general-purpose machines, *J. of Syst. & Software*, Vol.11, pp.181–198 (1990).
13) 朴 圭成，芦原 評，清水謙多郎，前川 守：分散オペレーティングシステムにおけるプロセス移送の方式，情報処理学会論文誌，Vol.31, No.7, pp.1080–1090 (1990).
14) 前川 守，所真理雄，清水謙多郎（編）：分散オペレーティングシステム，共立出版社 (1991).
15) 平野 聡，田沼 均，須崎有康：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，並列処理シンポジウムJSPP '93, pp.237–244 (1993).

付　　録

A.1 移送アルゴリズム概要

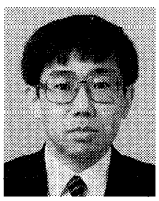
ここに示したプログラムでは，例として preorder でグラフをたどり移送起動処理を行うとした．異なる順序で移送起動処理をすることも可能である．また，各プロセスで次の移送先を計算し，その情報を事象の引数として渡すとした．すなわち移送起動指示とともに次に利用すべき移送先情報が渡され，移送起動終了とともに更新された移送先情報が渡される．移送エージェントが，プロセスを順に訪れることによって移送起動処理を行うと見なすこともできる．また，通信が必要になるが，移送管理サーバに問い合わせながら移送先を決定することも可能である．

InitiateSelfMigration は，自己プロセスの移送起動処理を始めることを意味する．すなわち，あらかじめ定められた順序に従って移送先を決め，そこへのデータ移動の準備等を行う．**SendEvent** は，第1引数で指定されるプロセス（群）に対して，事象を送信することを意味する．**SelectNext** は，次の移送起動対象を選択する．すべての移送起動処理が終了している場合には0を返すとする．

```
Process(Event event, Node from, Node next)
{
  extern Node  Manager;  // 移送管理プロセス
  static State state = NotMigrated;
  Node         to;
  List         parents;  // 移送起動指示送付元
  List         peers;    // 現在の通信相手
  List         toMove;   // 移送起動処理すべき相手

  switch ( event ) {
    case Connected:       // 接続
      peers.append(from); // リストに追加
      break;
    case Disconnected:    // 切断
      peers.remove(from); // リストから削除
      break;
  }
  switch ( state ) {
    case NotMigrated: // 未移送
      switch ( event ) {
        case Initiate:     // 移送起動指示
          parents.append(from);
          toMove = peers;
          state = InProgress; // 移送中
          InitiateSelfMigration(next++);
          SendEvent(self, Finished, next)
          break;
      }
      break;
    case InProgress:  // 移送起動処理中
      switch ( event ) {
        case Finished:     // 移送起動終了
          toMove.remove(from);
          to = SelectNext(toMove); // 次を選択
          if ( to ) { // 移送を指示
              SendEvent(to, Initiate, next);
          } else { // 移送起動処理終了
              state = Completed;
              SendEvent(parents, Finished, next);
          }
          break;
        case Disonnected:  // 切断
          // 移送管理プロセスに登録
          SendEvent(Manager, Connected, from);
          break;
        case Initiate:     // 移送起動指示
          parents.append(from);
          break;
      }
      break;
    case Completed:   // 移送起動処理完了
      switch ( event ) {
        case Connected:    // 接続
          // 移送管理プロセスに登録
          SendEvent(Manager, Connected, from);
          break;
        case Reset:        // リセット
          state = NotMigrated; // 非移送状態
          break;
      }
      break;
  }
}
```

(平成 9 年 5 月 19 日受付)
(平成 10 年 5 月 8 日採録)



大澤　範高（正会員）

昭和 58 年東京大学理学部情報科学科卒業．昭和 60 年同大学大学院理学系研究科情報科学専攻修士課程修了．昭和 63 年同博士課程修了．ソフトウェア開発会社を経て，平成 5 年電気通信大学大学院情報システム学研究科助手．平成 10 年メディア教育開発センター助教授．理学博士．並列分散システムソフトウェアに興味を持つ．ACM, IEEE-CS 各会員．

弓場　敏嗣（正会員）

昭和16年9月22日生まれ．昭和41年3月神戸大学大学院工学研究科修士課程修了．野村総合研究所を経て，昭和42年通商産業省工業技術院電気試験所（現，電子技術総合研究所）に入所．以来，計算機のオペレーティングシステム，見出し探索アルゴリズム，データベースマシン，データ駆動型並列計算機等の研究に従事．その間，計算機方式研究室長，情報アーキテクチャ部長等を歴任．平成5年4月より，電気通信大学大学院情報システム学研究科教授．並列処理一般に興味を持つ．工学博士．電子情報通信学会，日本ロボット学会，日本ソフトウェア科学会，ACM，IEEE-CS各会員．