システムソフトウェアと
オペレーティング・システム 83－3
（2000．2．24）

# クラスタにおける並列プロセスマイグレーションの評価

西岡 利博　　堀 敦史
エム・アール・アイ システムズ　新情報処理開発機構
{nishioka,hori}@trc.rwcp.or.jp

本論文は，クラスタ上での並列プロセスマイグレーションの実現方式と，その評価について述べる．並列計算を実行するプロセス群（並列プロセス）を，移送元のノードから移送先のノードに一斉に移送する並列プロセスマイグレーションでは，本来並列 OS によって禁止されなければならない，異なる並列プロセスへの通信が必要となる．これを効率良く実現するための手法を提案する．ギガビットネットワークによって接続された PC クラスタ上でこの方式を評価したところ，ノードあたりのデータ転送性能は 32 MB/s 程度であった．これは，同一ノード上のプロセス間のパイプの転送性能，および，そのクラスタのネットワークの転送性能のおよそ 60% に相当する．実並列プロセスのマイグレーションでは，移送先で転送されたメモリイメージを復元する処理のため，ノードあたり 13 MB/s 程度の性能となることが分かった．

# Evalution of Parallel Process Migration on a Cluster

Toshihiro Nishioka　　Atsushi Hori
M.R.I. Systems　Real World Computing Partnership
{nishioka, hori}@trc.rwcp.or.jp

This paper describes an implementation of a parallel process migration mechanism and its evaluations on a cluster. In parallel process migration, a group of processes executing a parallel computation must be moved to the other nodes at the same time. A special inter-node communication is required for the migration because normal inter-node communication is only permitted by parallel OS within the nodes where a parallel process is running. It is proposed to implement the protected inter-node communication in an efficient way. It has been evaluated on a PC cluster with a giga-bit network, and the 32 MB/s bandwidth per node was obtained, which corresponds to almost 60% of data transfer performance of a pipe connecting processes in a same node and data transfer performance of the cluster's network. Performance of migrating a real parallel application is reduced to 13 MB/s because the overhead of restoring the process' memory image.

## 1　はじめに

並列計算機やクラスタでは空間分割スケジューリング，つまり計算ノードの部分集合に分割して同時に複数のジョブを実行するように運用される場合が多い．このような空間分割スケジューリングにおいては，ユーザ並列プロセス[1]が投入されると，それが実行される部分集合を，システム全体の負荷が均衡するように割り当てることが可能であるが，並列プロセスの終了により負荷の均衡が崩れる場合がある．もし，負荷の高い部分集合にある並列プロセスを，負荷の低い部分集合にマイグレーション（移送）することができれ

[1]本論文では，クラスタにおける並列計算のために互いに通信し合う Unix プロセスの集合を「並列プロセス」と呼ぶ．

ば，負荷の不均衡を解消できる．また，ノード故障が発生した部分集合にある並列プロセスにおいては，直前のチェックポイントした状態から故障していない部分集合へマイグレーションすることで，実行を継続することが可能になる．このように並列プロセスのマイグレーションには利点がある．

我々は既にクラスタ上の並列OS SCore-D 上に並列プロセスのチェックポイント機構を実現した [10]. SCore-D は空間分割スケジューリングに加え，ギャングスケジューリングによる効率的な時分割スケジューリングを実現している．また，ユーザ並列プロセスにおいて効率的なユーザレベル通信を許すことから，ユーザ並列プロセスの通信状態のメモリへの退避／復帰をギャングスケジューリング時に行なうネットワークプリエンプション機構 [8, 3] を持っている．SCore-D のチェックポイントは，このネットワークプリエンプション機構を応用したものである [10].

チェックポイントは時間軸でのマイグレーションと考えることもできるため，原理的には SCore-D 上のチェックポイント機構を応用することで，並列プロセスのマイグレーションを実現可能である．並列 OS の立場から，個々のユーザ並列プロセスは互いに保護されている必要がある．つまり，ある並列プロセスが発した通信メッセージが他の並列プロセスへ到達できてはならない．しかし，並列プロセスのマイグレーションにおいては，移送元から移送先への通信が必要であり，この通信は保護の範囲を越えている．このため並列プロセスのマイグレーションにおいては，通信の保護をどのように実現するか，保護を実現することによる効率低下をいかに抑えるかが問題となる．

本稿は，SCore-D のようなクラスタを対象とした並列 OS において，保護された通信による並列プロセスマイグレーションを実現するための一手法を提案する．同時にこの手法を用いて実現された並列マイグレーションの性能の評価を行なう．チェックポイントの実装の詳細については文献 [10] を参照されたい．

## 2 SCore-D の概要

SCore-D[8, 3] は Unix 的にはデーモンプロセスの集合として実現される一種のミドルウェアであり，クラスタ上で走る全てのユーザ並列プロセスは SCore-D により生成され管理される．SCore-D は低レベルの通信レイヤーとして PM[6] を想定している．基本的に PM はノード単位の通信資源を管理し，SCore-D はクラスタ全体の資源管理を行なう．

PM はチャネルと呼ばれる仮想ネットワークを提供する [6]．これは物理ネットワークをソフトウェア的に多重化したものである．SCore-D は初期化時にクラスタが保持する PM デバイスをオープンし，ギャングスケジューリングのための同期や大域資源管理のために PM のチャネルをひとつ占有する．PM デバイスをオープンしたプロセスだけが PM の操作に対し特権を持つことが許される．ユーザ並列プロセスには，PM チャネルが並列プロセス生成時に必要な数だけ割り当てられ，終了時には割り当てられた通信資源は回収される．生成時にユーザ並列プロセスに割り当てられた仮想ネットワークにおいて，割り当てられたノードの集合を越えた通信は禁止される．この結果，通信の保護が実現される．

SCore-D ではユーザ並列プロセスに通信ハードウェアを直接操作することを許し，高性能な通信を実現している．このため並列プロセスを切替える際，通信ハードウェアの状態を退避／復帰する必要がある．通信ハードウェアの状態を含む PM チャネルの状態はコンテキストと呼ばれ，SCore-D はギャングスケジューリングのための全体同期と，ネットワーク中にメッセージが存在しないことを確認するための同期を行ない，コンテキストを切替えることで，並列プロセス切替を実現している．コンテキスト切替は，多くの場合チャネルとコンテキストを結ぶポインタの付け変えで済むため，高速なギャングスケジューリングが実現されている [8, 3]．並列プロセスの無矛盾なチェックポイントは，通信コンテキストとプロセスのメモリイメージをディスクに保存することを除いて，基本的にコンテキスト切替と同じである．我々は既に SCore-D 上にユーザレベルのチェックポイント機能を実装している [10].

SCore-D 自体はユーザレベルマルチスレッド言語 MPC++[4] で書かれている．これは Unix などの OS のカーネルが基本的にマルチスレッドで動作するため，マルチスレッドで記述した方が楽であるという理由による．ユーザ並列プロセスの強制終了など，イベントの処理はスレッドが生成され，そこで処理される．しかしながらユーザレベルスレッドであるため，ブロックするシステムコールを発行することができない．SCore-D 全体がブロックする可能性があるからである．従って，ブロックする可能性のある I/O はノンブロックとする必要がある．これは後述するような問題を生じる可能性がある．

## 3 マイグレーションの設計方針

本稿におけるマイグレーションの設計において留意した点は，*i)* 通信の保護を確保すること，*ii)* 高いマイグレーションのバンド幅を確保すること．*iii)*

チェックポイントとマイグレーションで出来る限りコードを共有すること，の3点である．

空間軸のチェックポイントと考えることが出来るマイグレーションは，チェックポイントにおけるディスクへの退避を，通信に置き換えることで実現可能となる．並列プロセスのマイグレーションを実現するには，走行中の並列プロセスに割り当てられているノード集合とは別なノード集合に，チャネルのコンテキストとプロセスコンテキストを移送しなければならない．しかしながら割り当てられたノード集合以外への通信は保護されているため，なんらかの特別な通信手段を用意する必要がある．

近代的なOSにおいて，保護された領域を越えて通信する場合は，OS経由で行なうことが原則になっている．本稿においても例外でなく，マイグレーションのための通信をSCore-D経由とすることで，保護を実現することを考える．

## 4 評価実装環境

以下，本稿における実装および評価は，PCクラスタであるRWC PCC-II[7]（表1）を用いた．

表1: RWC PCC-IIの仕様

| # of Nodes | 128 |
|---|---|
| Network | Myrinet<br>Fast Ethernet |
| Node Processor | PentiumPro<br>Clock: 200 [MHz]<br>Memory: 256 [MB]<br>Node OS: Linux |

マイグレーションの設計に関係すると思われる，本クラスタにおけるいくつかの通信性能を計測した結果を表2に示す．ここでは，ノード間通信に加え，ノード内のプロセス間通信についても計測した．この表における PM の最高バンド幅は Myrinet NIC の DMA を直接用いたゼロコピー通信を用いた値であり，PCI バスの上限（133MB/s）に近い値となっている．MPC++ の通信は PM を用いている．MPC++ の最高バンド幅が Myrinet の約半分以下となっているのは，ゼロコピー通信を用いていないため，bcopy() のオーバヘッドがボトルネックとなっているからである．PentiumPro（200MHz）における親子プロセス間の Unix pipe の最高バンド幅は MPC++ のバンド幅と同等である．ちなみに Pentium-III（500MHz）における MPC++ と Unix pipe の最高バンド幅は共に 100 MB/s を越える．

表2: RWC PCC-II の通信性能

| Myrinet | Max. Bandwidth: 119 [MB/s]<br>Min. Latency: 9.25 [us] |
|---|---|
| MPC++ | Max. Bandwidth: 49.6 [MB/s] |
| Unix pipe | Max. Bandwidth: 52.6 [MB/s] |

マイグレーションの評価には，NAS 並列ベンチマークプログラム [1]（NPB）を用いた．

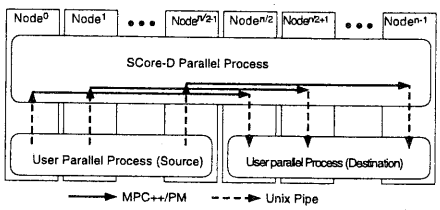


図 1: マイグレーションにおける通信の発生箇所

## 5 通信方式

図1は並列プロセスのマイグレーションで発生する通信を示したものである．この図では $n$ 台のノードで構成されるクラスタにおいて，前半分で動作している並列プロセスを後半分に移送する例である．通信は3箇所で発生する．ユーザプロセスのメモリイメージをSCore-Dに送るプロセス間通信，SCore-Dが移送先に転送するノード間通信，そして，移送先の個々のノードにおいてSCore-Dから転送先のプロセスにメモリイメージを送るプロセス間通信である．実際にはSCore-Dが保持する通信のコンテキストの移送があるが，プロセスのメモリイメージに比べ量的に少ない（1 MB以下）ため省略した．

最初の実装では，SCore-Dとユーザプロセス間の通信にUnixのpipeを用い，ノード間通信はMPC++の遠隔メモリアクセス機能[4]を用いた．こうすることで3箇所の通信のバンド幅がほぼ50MB/s（表2）となり，ボトルネックが生じないと考えたからである．さらに，チェックポイント時には，SCore-DがpipeのデータをファイルにSCore-Dが書き出すだけで済む．つまり，SCore-Dとユーザプロセス間をpipeで接続することで，ユーザプロセス側のチェックポイントライブラリはチェックポイントもマイグレーションも全く同じコードとすることが可能となる．

前述したようにSCore-D内で発行するシステムコールでブロックすることはできない．ただしselect()によるブロックは例外である．何故ならSCore-DはPMのメッセージ受信をselect()で待っているからである．そこで，ユーザプロセスからメモリイメージを受けとるためのpipeからのread()はselect()で待つこととし，逆のpipeへの書き込みはノンブロックモードでのwrite()とした．ノンブロックによるwrite()がEAGAINで失敗した場合，書込量を半分にし再試行するという処理を繰り返すようにした．

この方式を実装し，評価したところ，マイグレーション全体のバンド幅（定義については第7.1節を参照）は，高々数MB/sという低い値しか得られなかった．この状況はダブルバッファの手法を組み込んでも根本的な改善には至らなかった．その直接的な原因は解明されていないが，表2の結果から，select()によるread()の待ちとノンブロックによるwrite()の待ちがうまく噛み合わなく，データの流れが滞ったものと推測される．

## 6 改良方式

マイグレーションの性能を改善するために別な実装方式を考える．問題がselect()による待ちやノンブロックのwrite()にあるとすれば，pipeに対するread()やwrite()を通常のブロックするモードで行なえば良いことになる．SCore-D内でブロックすることができないのならば，SCore-Dの外でブロックすれば良いことになる．そこで，SCore-Dが本来やるべきマイグレーションに関する通信を，SCore-Dが新たにfork()した別の子プロセスにSCore-Dの代理としてやらせる（図2）．メモリイメージを移送先に転送するために，マイグレーション専用のPMチャネルを事前にSCore-Dが確保しておき，代理子プロセスが用いる．こうすることでSCore-Dの通信とは独立にマイグレーションのための通信が可能になる．代理プロセスはSCore-Dプログラムなので，この方式においても通信の保護は実現可能である．

SCore-Dが他のプロセスをスケジューリングするために子プロセスを止める（SIGSTOPを配送する）際には，この代理子プロセスに対しても同じようにシグナルを配送すれば良い．SCore-Dはマイグレーション専用のPMチャネルを資源として管理し，別のユーザ並列プロセスが同時にマイグレーション専用チャネルを使わせないようにする．

この実装は最初の実装に比べ，read()やwrite()がブロックした時の処理を必要としないばかりか，複雑なダブルバッファの処理も不要になるため，マイグレーションのためのプログラムも単純化されるという利点も見い出すことができた．次章において，本方式の実装における評価結果を示す．

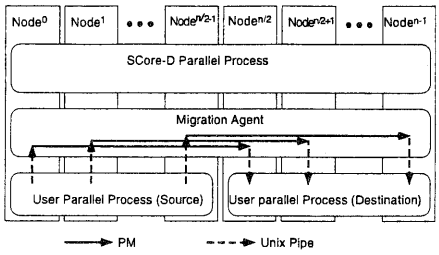


図2: 改良方式における通信

## 7 評価

評価は，評価環境上で，評価用のプログラムを移送させることで行った．

### 7.1 性能指標および計測方法

性能指標として，下式の値を使用する．

$$b = \Sigma_{i=0}^{n-1} l_i/n/T_{max}$$

ここで，$n$はノード数，$l_i$は，ノード$i$が転送したメモリイメージの大きさ，$T_{max}$は，各移送先で記録した消費時間のうちの最大の値である．この値$b$を「バンド幅」と呼ぶ．単位はMB/sである．$\Sigma_{i=0}^{n-1} l_i/n$は，各ノードが転送したメモリイメージの大きさの平均値なので，「ノードあたり転送サイズ」と呼ぶ．

本マイグレーション機構では，移送先のプロセスが実行を開始すると，SCore-Dに移送の開始を要求する．移送先のすべてのプロセスで移送開始の要求がなされると，移送元のプロセスの停止とデータの転送が開始される．そこで，消費時間として，移送先のプロセスでSCore-Dに移送開始の要求を出す直前から，ユーザプログラムの実行を再開する直前までを計測する．

### 7.2 評価用プログラム

評価には2種類のプログラムを用意した．ひとつは，代理子プロセス方式のデータ転送性能を評価するためのテストプログラムである．このテストプログラムは，マイグレーションが指示されると，メモリイ

メージの代わりに，予め用意したバッファの内容を，指定された回数だけ繰り返し転送する．移送先ではそれを読み込むが，それを用いてメモリイメージを復元することはしない．消費時間として，移送先のプロセスで SCore-D に移送開始の要求を出す直前から，転送されてきたデータの読み込みを終了するまでの時間を計測する．

このテストプログラムを複数のノードで実行すると，すべてのノードが同じ量のデータを転送する．このテストプログラムでは，実際のマイグレーションから，転送するメモリイメージを準備する処理と，転送されてきたメモリイメージを復元する処理が省略されており，これによって得られた値が，本マイグレーション機構の性能の上限値と考えられる．

もうひとつは，実アプリケーションとして NPB の中から CG および IS を選択した．NPB はプログラム毎に問題の大きさのクラスが設定されている．今回はクラス A とクラス B を用いた．クラス B はクラス A よりも大きい．異なるクラスのプログラムをマイグレーションすることで，転送量とバンド幅の関係を調べることができる．

## 7.3 計測結果

図 3は，テストプログラムによる計測結果である．横軸はノードあたり転送サイズ，縦軸はバンド幅である．ノード数は 4 とし，転送データ量を 1 MB ～ 128 MB の範囲で変え，それぞれ 10 回ずつ計測した．

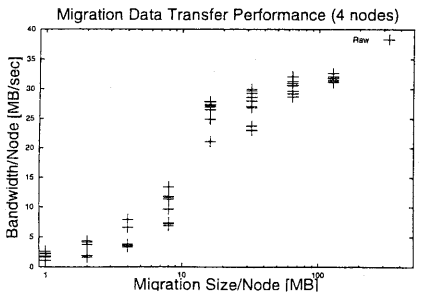


図 3: テストプログラムの計測結果（4 ノード）

転送データ量が多くなるほど性能が良くなるのは，データ転送以外のオーバヘッドがあるためと考えられる．このオーバヘッドには，代理子プロセスの起動，および，通信コンテキストの転送が含まれている．

表 3は，NPB から CG のクラス A のプログラムを，さまざまなノード数で実行し，それをマイグレーションして得られた計測結果である．

表中，プログラム名を表す CG.A.4 とは，アルゴリズム CG を，クラス A，ノード数 4 で実行するプログラムという意味である．

表 3: さまざまなノード数での計測結果

| プログラム | $T_{max}$ | $\Sigma_{i=0}^{n-1} l_i/n$ | $b$ |
|---|---|---|---|
| CG.A.4 | 1.07 (sec.) | 14.1 (MB) | 13.1 (MB/s) |
| CG.A.8 | 0.76 (sec.) | 7.26 (MB) | 9.61 (MB/s) |
| CG.A.16 | 0.61 (sec.) | 3.75 (MB) | 6.11 (MB/s) |
| CG.A.32 | 0.37 (sec.) | 2.05 (MB) | 5.60 (MB/s) |

ノード数が少なくなると，ノードあたりの転送サイズが大きくなるので，少ないノード数のプログラムの方が良い性能を示す．NPB プログラムを 4 ノードで実行してマイグレーションした場合の計測結果を，図 4に示す．それぞれ 10 回ずつ計測した．

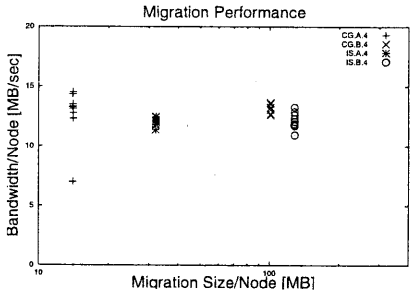


図 4: NPB の計測結果（4 ノード）

“Raw” の性能が，128 MB を転送する場合でおよそ 32 MB/s であるのに対し，NPB を移送した場合の性能は，改良前の数 MB/s よりは改善されたとはいえ，13 MB/s 程度に留まっている．これは，代理子プロセスによるデータ転送以外の部分にボトルネックがあるものと考えられる．マイグレーションの際，データ転送以外に行われる処理は，送信元でメモリイメージを転送可能な状態に整える処理と，送信先でメモリイメージを復元する処理である．

これを調べるために，次の実験を行った．

送信元ノードで，メモリイメージを代理子プロセスに出力する代わりに /dev/null に出力し，その処理の消費時間からバンド幅を計算すると，IS, CG の 4 ノードのプログラムの場合で，およそ 30 MB/s であり，“Raw” の数値に比べて低くはない．

一方，送信元のノードで，直ちに転送可能なメモリイメージをあらかじめメモリ上に用意した状態からマイグレーションを行い，送信先のノードでメモリイメージを復元する処理の消費時間を計測してバンド幅を計算すると，およそ 13 MB/s である．このことから，送信先のノードでメモリイメージを復元する処理

がボトルネックになっていることが分かった.

文献 [9] で我々は，マイグレーションの通信パターンにおける PCC-II のネットワークのデータ転送性能を評価し，移送元のノードと移送先のノードに重複がない限り，どのような組合せであっても，また，いくつのノードが使用されていても，最低で約 16 MB/s の転送性能が得られることを調べた．現在のマイグレーションの性能はこの値に届いていないので，更に改良を加えて高速化することが今後の課題である.

## 8 関連研究

プロセスマイグレーションは，分散 / 並列計算環境における負荷平衡に必要な技法であり，多くの研究がなされてきた．しかし，これまでの研究の多くは，LAN で接続された分散計算環境でノード間の負荷を調整するために，並列プロセスの一部のプロセスを他のノードに移すタイプのものであり（例えば [5, 2]），調査した範囲では，本研究のように，並列計算環境上で，並列プロセス全体を移送する例は見られなかった.

## 9 まとめ

クラスタ上で，保護された通信による並列プロセスマイグレーションを実現する一手法を提案した.

ある並列プロセスの発した通信メッセージが他の並列プロセスへ到達することは，並列 OS によって禁止されている必要があるが，並列プロセスマイグレーションにおいては，移送元から移送先への通信が必要となる．このため，マイグレーションのための通信を SCore-D 経由で実現したが，素朴な実現では，転送性能の点で問題があった.

提案手法は，マイグレーションのための通信を，SCore-D が新たに fork() した別の子プロセスに代行させる方式とした.

この方式を PCC-II 上で評価したところ，データ転送性能としては，この機構での通信を実現している pipe およびネットワークの転送性能（いずれも，およそ 50 MB/s）から見て妥当な転送性能（およそ 32 MB/s）が確保できた．しかし，転送されたメモリイメージを移送先で復元する処理がボトルネックとなり，並列プロセスマイグレーション全体では 13 MB/s 程度の性能に留まった.

## 参考文献


[1] D. H. Bailey, J. T. Barton, T. A. Lasinski, and H. D. Simon. The NAS Parallel Benchmarks. NASA Technical Memorandum 103863, NASA Ames Research Center, 1993.

[2] A. Barak, O. La'adan, and A. Shiloh. Scalable Cluster Computing with MOSIX for LINUX. In *Linux Expo '99*, pp. 95–100, Raleigh, N.C., May 1999.

[3] A. Hori, H. Tezuka, and Y. Ishikawa. Highly Efficient Gang Scheduling Implementation. In *Supercomputing '98*, Nov. 1998.

[4] Y. Ishikawa. Multi Thread Template Library - MPC++ Version 2.0 Level 0 Document -. Technical Report TR-96012, RWC, September 1996.

[5] G. Stellner. CoCheck: Checkpointing and Process Migration for MPI. In *Proceedings of the 10th International Parallel Processing Symposium (IPPS '96)*, pp. 526–531, Honolulu, Hawaii, Apr. 1996.

[6] H. Tezuka, A. Hori, Y. Ishikawa, and M. Sato. PM: An Operating System Coordinated High Performance Communication Library. In *High-Performance Computing and Networking (HPCN '97)*, Vol. 1225 of *Lecture Notes in Computer Science*, pp. 708–717, Vienna, Austria, Apr. 1997. Springer-Verlag.

[7] 手塚, 堀, F. B. O'Carroll, 石川. RWC PC Cluster II の構築と性能評価. In *HOKKE '98*, Mar. 1998.

[8] 堀, 手塚, 石川. ギャングスケジューリングの高速化技法の提案. 並列処理シンポジウム JSPP '98, pp. 207–214, June 1998.

[9] 西岡, 堀, 手塚. ギガビットネットワークを用いた並列プロセスマイグレーションの性能評価. 情報処理学会研究報告 99-OS-134 (SWoPP'99), pp. 81–88, August 1999.

[10] 西岡, 堀, 手塚, 石川. クラスタにおけるコンシステントチェックポイントの実現. 並列処理シンポジウム JSPP'99, pp. 229–236. 情報処理学会, June 1999.