分散共有メモリモデルに基づくHPC環境の高性能実装と性能評価

大西淑雅[†] 了戒清[††,*] 末吉敏則[††]

分散スーパーコンピューティング環境(DSE)は，分散システム上に分散共有メモリモデルに基づく並列処理機能を実現するハイパフォーマンスコンピューティング環境である．DSEでは移植性を考慮して，既存OS(UNIX)には手を加えず，すべての処理をユーザレベルで実装する方針を採っている．しかし，ユーザレベルでの実装を行うが故に，DSEにおける並列処理は必ずUNIXカーネルを介して行われるため大きなオーバヘッドを伴い，効率良い並列処理を妨げていた．本稿では，分散共有メモリモデルに基づく並列処理に伴う細かな粒度の通信にも対応可能なハイパフォーマンスコンピューティング環境の実現を目標に，ユーザレベルで実装する際の通信処理を改善する高性能実装法を提示する．具体的には，通信処理における入出力関数の呼出しに着目し，従来は別々のUNIXプロセスで実現していたDSEのカーネルと応用プロセスの処理を1つのUNIXプロセス内で行うことによる通信処理の改善について説明する．また，並列処理プリミティブや並列アプリケーションを用いて，提示した高性能実装法の性能評価を行い，その有効性を確認した．そして，DSEの性能をさらに向上させるには，プロトコル処理などのネットワーク通信処理の高速化が重要であることが明らかにした．

Efficient Implementation and Evaluation of a High-Performance Computing Environment Based on a Distributed Shared Memory Model

YOSHIMASA OHNISHI,[†] KIYOSHI RYOUKAI[††,*] and TOSHINORI SUEYOSHI[††]

The Distributed Supercomputing Environment (DSE) provide a high-performance computing environment based on a distributed shared memory model. Our concept stipulates that all DSE processing shall be implemented at user-level. However, this implementation at user-level impedes the speed-up gained by parallel processing. This is because the DSE parallel processing is performed via the UNIX kernel which incurs a large overhead. This paper describe the implementation of a high-performance computing environment to execute small grain communication in parallel processing based on distributed shared memory model. In particular, we explain with emphasis on the input/output function calls in communication processing, and the efficient implementation method to improve the communication processing of DSE kernel and application process which were merged into one UNIX process in the present DSE. We estimate the efficiency of our implementation and confirm its effectivity by running both the primitives and some applications. Consequently, we verified the importance of speeding up the network communication processing, such as protocol processing, to further improve the DSE performance.

1. はじめに

最近では，ネットワークの高速化やマイクロプロセッサの高性能化に伴って，複数のワークステーションをLANで接続した分散システムが広く研究所や大学等で構築されている．さらに，分散システムに散在した計算機資源を効率良く利用しようとする研究が活発に行われている．その1つに分散システムの持つ資源を管理する能力および高速な通信機能を利用して，システムが内包する計算機上で並列処理を行う研究がある．このような分散システムを実現したものとしてV-system[1]，Chorus[2]などがある．しかし，これらの分散システムが提供する並列処理機能を利用するには，これまで利用してきた既存OSの変更もしくは更新が必要となり，ユーザが必要に応じて並列処理機能を気軽に利用することは難しい．

このような背景の中，現存のOS(UNIX)に手を入れ

† 九州工業大学情報科学センター
Information Science Center, Kyushu Institute of Technology
†† 九州工業大学情報工学部知能情報工学科
Department of Artificial Intelligence, Kyushu Institute of Technology
* 現在，(株)富士通
Presently with Fujitsu Corporation

ずに並列処理を実現する環境として，PVM[3] に代表されるメッセージ通信をベースとした研究や，共有メモリをユーザに提供するといった研究[4] が多くの研究者によって進められている．共有メモリをユーザに提供するといった研究では，ユーザのプログラムの記述性を容易とするメモリ R/W 方式を実装している．しかしメモリ R/W 方式は I/O インタフェース方式と比べて，その処理に大きな通信コストを要するため，イーサネットのような LAN をベースとした分散システムでは，その性能を十分に発揮できていないのが現状である．そこで，我々は既存の LAN を前提とする分散システム上でのハイパフォーマンスコンピューティング(HPC)環境の実現を重視し，少ない通信コストで実現できる I/O インタフェース方式の共有メモリを提供する分散スーパーコンピューティング環境(DSE)[5)~7)] の開発および研究を行っている．

DSE では，プロセッサ要素間の通信はグローバルメモリ(共有メモリ)を介して実現されるため，細かな粒度の通信にも対応できる並列処理機能の提供が要求される．しかし，DSE は UNIX 環境で気軽に利用できるように，移植性を考慮して，すべての処理をユーザレベルで実装する方針を採っている．そのため，DSE が行う通信は必ず UNIX カーネルを介す必要があり，通信処理に大きなコストを要することになる．その結果，我々の初期の実装方法[5] では，細かな粒度の通信にも対応できる並列処理機能の提供という要求を完全に満たすには至っていない．実際，過去の調査[8] からも，通信処理がシステムの性能に対して大きな影響を及ぼしていることが判明している．

本稿では，かなり細かい粒度の通信に対応可能な汎用性のある並列処理環境の実現を目標に，通信処理を改善するための高性能実装法を提案し，その有効性を明らかにする．以下，2 章では DSE の概要を述べる．また，3 章では DSE の通信処理の影響について述べた後，通信処理を改善する高性能実装法を説明する．さらに，4 章ではこの高性能実装法に対する性能評価を行い，最後にまとめを述べる．

2. 分散スーパーコンピューティング環境の概要

DSE は，UNIX オペレーティングシステムに変更を加えず，分散共有メモリモデルの並列処理環境をユーザに提供する．ユーザに共有メモリモデルの環境を提供することによって，プログラムの連続性を保証し，従来のアプリケーションプログラムの移植を容易にしている．

DSE のシステムモデルを図 1 に示す．DSE におけるプロセッサ要素(PE)はそれぞれ，分散システム上のワークステーションに対応している．各 PE はプロセッサユニット(PU)以外に，グローバルメモリ(GM)とローカルメモリ(LM)を保持する．グローバルメモリは共有メモリを構成し，単一のアドレス空間を形成している．また，DSE は UNIX 環境にユーザレベルで実現されているため，並列処理の単位となるプロセスは基本的には UNIX プロセスであり，UNIX がプロセス

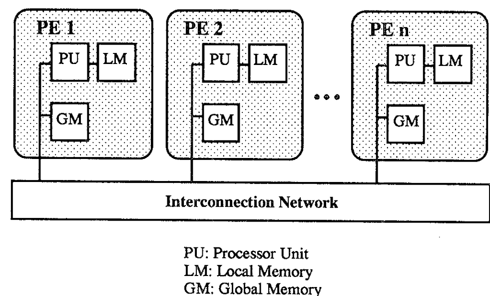


図 1 DSE のシステムモデル
Fig. 1 The DSE system model.

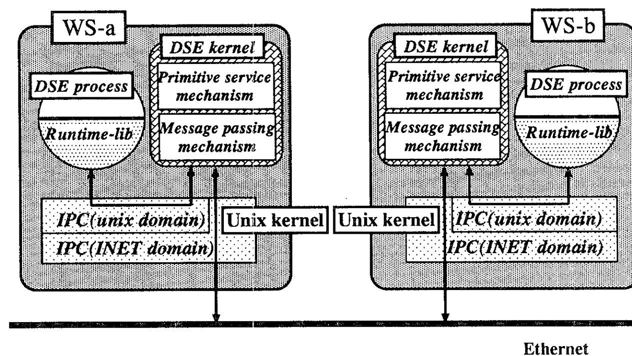


図 2 ソフトウェア構成
Fig. 2 The DSE software organization.

を生成する時に確保するメモリ空間をローカルメモリとみなす．よって，ローカルメモリの場合，同一ワークステーション内であっても，異なるプロセッサ要素間ではローカルメモリを介したデータ共有はできない．

DSE のソフトウェア構成を図 2 に示す．DSE における 1 つのプロセッサ要素は，DSE カーネルと DSE プロセスの 2 つを 1 組として構成されており，それぞれ異なった UNIX プロセスで実現している．DSE カーネルは並列処理のための基本機能である DSE プリミティブ[5] の処理を行い，実質的にアプリケーションに並列処理機能を提供する部分であり，もう一方の DSE プロセスは並列アプリケーションを実行するための空間を提供している．

共有メモリは各プロセッサ要素ごとに分散配置されており，各々の DSE カーネルによって管理される．共有メモリの論理アドレス空間は，プロセッサ要素の番号(16 bit)とプロセッサ要素内のアドレス(32 bit)の 48 bit で構成されており，ユーザはこの論理アドレス空間を利用することができる．しかし，1 つの DSE カーネルによって物理的に確保されている共有メモリ空間は，DSE の基盤として使用する分散システム(ワークステーション)の能力に合わせて異なる．例えば，1 つの DSE カーネルによって物理的に確保されている共有メモリ空間が 256 K バイト(1 ページ 4 K バイトを 64 ページ持つ)に設定されている場合，ユーザは 32 bit で指定されるアドレスのどの部分でもアクセスできるが，64 ページを越えるアクセスを行うことはできない．こうすることで使用する分散システムの違いが，DSE の共有メモリの論理アドレス空間に影響を与えないようになっている．

論理アドレスから物理アドレスへの変換は 2 段階に分けられる．まず，論理アドレスのプロセッサ番号(プロセッサ要素の番号)部に指定されているプロセッサ要素の DSE カーネルへアクセス要求メッセージが送られる．次に要求を受けた DSE カーネルは，プロセッサ要素内のアドレスをアドレス変換機構によって物理アドレスに変換を行うことでアドレス変換を終える．

以上，DSE の実現メカニズムを簡単に述べた．次に DSE の利用方法について述べる．ユーザは DSE がライブラリとして提供する並列処理のための関数群(並列分岐，同期制御，共有メモリのアクセス)を利用し並列プログラムを記述する．この時，並列性の抽出，並列処理単位のプロセッサ要素への割り付け，およびデータの割り付け(共有メモリへの割り付け)に関してはユーザがすべて決定し記述する必要がある．

次に，DSE 上で並列アプリケーションを実行する前準備として，プロセッサ要素を分散システム上のワークステーションにどのように割り付けるかを記述する．通常，1 つのプロセッサ要素に対し 1 つのワークステーションを対応させるが，プロセッサ要素の数に対してワークステーション数が不足する場合は，1 つのワークステーションに複数のプロセッサ要素を割り付けることもできる．なお，この場合，それぞれのプロセッサ要素は異なるプロセッサ番号が与えられている．

DSE の起動は支援ツールを使うことによって 1 台のワークステーションから操作を行うことができる．支援ツールでは，先のプロセッサ要素の割り付け情報を参照して，各ワークステーション上に必要なプロセスを起動し，プロセッサ要素同士の通信路を確保する．この状態で，DSE はコマンド待ち状態となり(DSE シェルの起動)，並列プログラムを実行することができる．

以上，DSE の概要について簡単に述べた．なお，DSE の詳細な利用方法および実装方法などについて参考文献 5)～7) を参照されたい．

3．通信処理の影響と高性能実装法

DSE の実装において，通信処理は DSE プリミティブの処理速度を直接左右する要因である．ここでは，まず通信処理の影響について簡単に述べた後，通信処理を改善する高性能実装法について説明する．

3.1　通信処理の影響

通信処理が DSE に与える影響を調べるために，DSE における各機能の処理時間を測定した[8]．DSE が行う処理を通信処理と DSE プリミティブのための処理に分けてその処理時間を比較した場合，その比率はほぼ 9：1 であることが明らかとなった．さらに，その通信処理は read()，write()，select() 等の入出力システムコールの処理と UNIX カーネルが行うプロトコル処理などのネットワーク通信処理とに分けることができ，これらの処理時間比はおよそ 2：1 であることも分かった．つまり，入出力システムコールの処理時間は結果的に全体の 6 割にも及び，DSE の処理効率に重大な影響を与えていることが判明した．

以上の調査結果から，DSE プリミティブのための処理に関する部分を改善するよりも，通信処理に関する部分の改善の方が，その処理が全体のほぼ 9 割を占めることから考えて，大幅な処理の高速化を期待できる．しかしながら，通信処理のうち，プロトコル処理などのネットワーク通信処理は大半を UNIX カーネルに

依存しているため，ユーザレベルでの処理の改善は難しい．一方，処理全体に対して大きな割合を占める入出力システムコールの呼出回数の削減による処理の高速化については，その処理がUNIXカーネルに依存しないため，ユーザレベルで改善を行うことが可能である．

3.2 高性能実装法

我々の初期のDSE実装では，DSEカーネルとDSEプロセスの2つの処理の流れを独立に処理するため，2つの異なったUNIXプロセスを用いて実現している．そのため，DSEプロセス-DSEカーネル間およびDSEカーネル-DSEカーネル間の通信の際には，必ず入出力システムコールが使用されることになる．

そこで，本研究ではDSEカーネルとDSEプロセスの2つの処理を1つのUNIXプロセス上に実装することによって，DSEプロセス-DSEカーネル間の入出力システムコールの削除を図ることにした．しかし，この実装法を実現するには，次の2つの問題に対処する必要がある．

（1） アプリケーションプログラムのロード

最初の問題は，並列アプリケーションプログラムのロードをどのように実現するかである．DSEは様々な並列アプリケーションの実行環境を提供するものであるため，DSE起動後にアプリケーションプログラムをDSEプロセスにロードする必要がある．初期のDSE実装では，並列アプリケーションプログラムをDSEカーネルとは異なるUNIXプロセス(DSEプロセス)に，図3(a)のようにexec()関数を用いて，簡単にロードを実現していた．しかし，DSEカーネルとDSEプロセスの別々の処理を同一のUNIXプロセス空間に写像するには，exec()関数によるアプリケーションプログラムのロードは不可能となる．

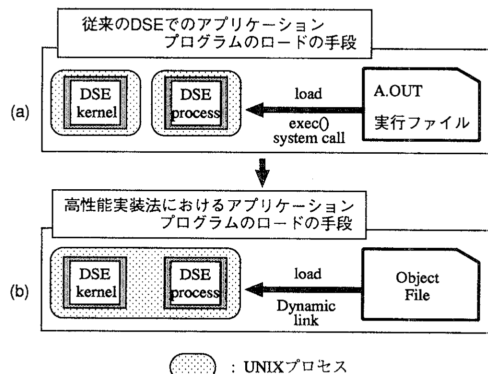


図3 アプリケーションプログラムのロード
Fig. 3 The model of loading application program.

そこで，DSEカーネルとDSEプロセスを1つのUNIXプロセスに実装するために，ダイナミックリンク[9),10)]を用いた．図3(b)のようにダイナミックリングを使うことで，1つのUNIXプロセスにおいても，動的に並列アプリケーションをDSEにロードすることが可能となる．

（2） DSEカーネルとDSEプロセスの独立性

二番目の問題は，DSEカーネルとDSEプロセスの2つの処理を1つのUNIXプロセスで独立に行う点である．単に2つの処理を独立に扱うだけであれば，setjmp()/long jmp()関数を用いた簡単なコルーチンで処理が可能である．しかし，この手法ではシグナル割込みを用いたコンテクスト切替えは難しい．DSEの処理では，任意の時刻に到着したメッセージの処理を行わなければならない．このため，シグナル割り込みを用いた横取り可能なスケジューリングをサポートする必要がある．

この問題の解決策としては，ユーザレベルで実装させたスレッドを利用する方法と，シグナルハンドラにDSEカーネルの処理を実装する方法がある．そこで今回，スレッド(SUN Microsystems社のLightweight Process Library (LWP lib)[10)]による方法と，シグナルによる方法の両方について，DSEの実装を行ってみた．

スレッドによるDSEは，DSEプロセス，DSEカーネル，およびMessage Catcherの3つのスレッドから構成される．Message Catcherは常にポーリングしており，他のプロセッサ要素のDSEカーネルから非同期で到着するメッセージを受け付け，DSEカーネルへスレッド間通信を利用してメッセージの到着を知らせる．このような構成を採ることで横取り可能なスケジューリングを実現している．

一方，シグナルを用いたDSEは，非同期入出力モー

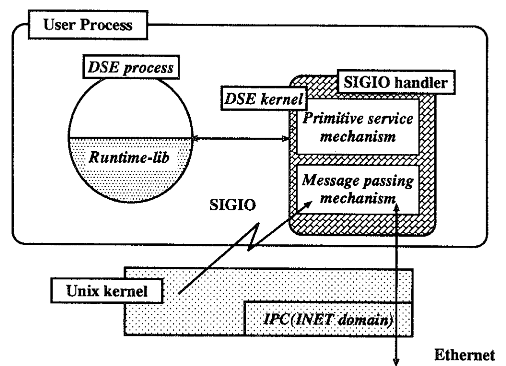


図4 シグナルを用いた実装例
Fig. 4 Implementation of Signal-based DSE.

ドにおける SIGIO の割込みを利用する．この手法では，ソケットへ入力が生じた場合に SIGIO の割り込みが生じ，入力があったことを知ることができる．この割込み中に DSE カーネル処理を行うことにより，任意の時刻に到着したメッセージにも対応できる構成となる．図 4 にシグナルを用いて実装を行った DSE の構成を示す．

4. 性能評価

スレッドならびにシグナルを用いて新たに実装を行った DSE について，我々の初期の実装法に基づく DSE による実行結果と比較し，性能評価を行った．ここで，各々の実装に基づく DSE を区別するために名称を付けることにする．つまり，初期の実装法に基づく DSE を Unix process-based DSE，高性能実装法に基づく DSE のうちスレッドによるものを Thread-based DSE，またシグナルによるものを Signal-based DSE と呼ぶ．

性能評価は，2 段階で行った．最初にまず，DSE プリミティブ単独について実行時間を測定し，各々の高性能実装法に関してプリミティブ実行における改善を調べた．次いで，DSE プリミティブに関する性能評価を参考にして，いくつかの並列アプリケーションについて実行時間を測定し，アプリケーションの処理効率に及ぼす影響を調査した．なお，測定環境としては SPARC station 2 から成る分散システムを使用した．

4.1 DSE プリミティブによる性能評価

DSE プリミティブのうち，グローバルメモリからデータを読み出す READ プリミティブの処理時間について測定を行った．アクセスすべきグローバルメモリの所在によって，自プロセッサ内のグローバルメモリにアクセスする場合と，他プロセッサ内のグローバルメモリにアクセスする場合とに分けて測定した．測定

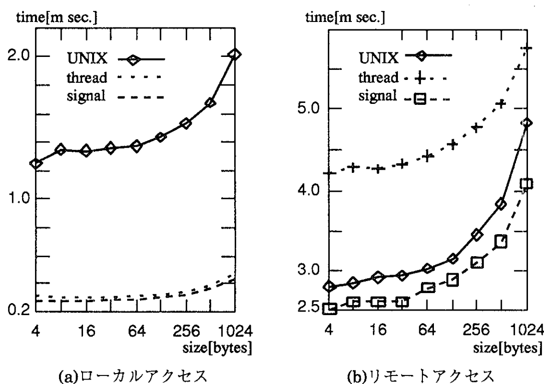


図 5　READ プリミティブの実行時間
Fig. 5　Execution time of READ primitive.

結果を図 5 に示す．

（a）　ローカルアクセスの場合

自プロセッサ内のグローバルメモリにアクセスした場合，Unix process-based DSE では DSE プロセス-DSE カーネル間で UNIX カーネルを介したプロセス間通信が必ず行われるが，高性能実装法ではプロセス間通信は生じない．そのため，図 5（a）に示すように Thread-based DSE，Signal-based DSE 共に大幅な改善が見られた．これからも，初期の実装法では通信処理が DSE の速度向上を大きく妨げていたことが分かる．

また，Unix process-based DSE は通信機構をソケットを用いて実現しているので，データのコピー処理が重く，プリミティブ処理における通信処理の割合が高くなる原因となっている．これに対し，Thread-based DSE，Signal-based DSE では共に，DSE プロセス-DSE カーネル間の通信はメッセージポインタの交換で行われているため，非常に軽い通信機構を実現している．このため，読み出すデータ量が増えても，READ プリミティブの処理時間はそれほど増加していない．

（b）　リモートアクセスの場合

ネットワークを介して他プロセッサ内のグローバルメモリにアクセスする場合は，プロセス間通信の実行回数を減らすことはできても，完全に無くすことはできない．よって，図 5（b）に示すように，ローカルアクセスの場合に見られたような飛躍的な改善は認められない．むしろ，Thread-based DSE の場合においては，処理速度が逆に遅くなっている．

Thread-based DSE では，標準の I/O ライブラリではなく，他のスレッドの処理をブロックせずに入出力処理を行う非閉塞 I/O ライブラリを用いて入出力処理を実現している．しかし，非閉塞 I/O ライブラリ関数の処理時間は標準の I/O ライブラリ関数に比べて遅く，また I/O イベント待ちのスレッドに対するコンテクスト切替えに非常に大きな時間を要する（UNIX システム（Sun）が提供するライブラリのため詳細は不明である）．その結果，UNIX カーネルに対して入出力処理を行う必要のあるリモートアクセスの処理においては，かえって大きな処理の遅滞を招くこととなった．

一方，Signal-based DSE ではローカルアクセスの場合と同様に，DSE プロセス-DSE カーネル間の通信処理が削減された効果が認められた．この効果は，メッセージサイズ（共有メモリへのアクセスサイズに比例する）が大きくなるに従っていっそう顕著になっている．これは，DSE プロセス-DSE カーネル間の通信処

表 1 偏微分方程式での実行時間
Table 1 Execution time of Partial Differential Equation.

格子点数	Unix process-based DSE (秒)	Signal-based DSE (秒)	改善率
16×16	4.458	2.727	1.635
32×32	8.974	5.641	1.586
48×48	14.273	9.196	1.552
64×64	21.606	14.203	1.521
96×96	39.204	30.588	1.282
128×128	69.368	54.134	1.281
256×256	323.805	303.515	1.067

理に使用されていた read()/write() システムコールの処理時間がデータ量に比例するためである．

4.2 並列アプリケーションによる性能評価

前節で処理速度の改善が認められた Signal-based DSE と初期の実装法に基づく Unix process-based DSE を用いて，実際の並列アプリケーションによる高性能実装法の性能評価を行った．性能評価に用いた並列アプリケーションは，偏微分方程式の求解，探索問題を扱う騎士巡回問題，およびマンデルブロー集合の描画計算の 3 つである．

（1） 偏微分方程式

偏微分方程式の数値解を SOR 法を用いて求める問題である．並列処理を行うために，領域を離散化して複数のブロックに分割し，それぞれを各々のプロセッサに割り当てる．各プロセッサは割り当てられたブロック内の格子点について，他プロセッサと協調動作を行いながら解を求めていく．このアプリケーションでは，格子点の粗さ(格子点数)を調整することによって並列計算の粒度を制御できる．なお，並列処理に使用したワークステーションは 4 台である．

表 1 に，Unix process-based DSE および Signal-based DSE 上で偏微分方程式を解くために要した実行時間と，Unix process-based DSE を基準とした場合の改善率を示す．ここで，偏微分方程式の粒度を計算と通信の比率を用いて[格子点数(計算数)/通信数]として表現する．[10/1]は，10 回の計算ごとに 1 回の通信を行うことを意味する．並列処理の粒度が小さくなる([256×256/1]->[16×16/1])に伴って改善率が向上している．このことは，高性能実装法に基づく DSE が我々の初期の実装法よりも細かな通信粒度あるいは計算粒度となる問題にも対応でき，適用できる応用範囲が広がったことを意味している．当然ながら，格子点数が 256×256 のように多くなる場合には，改善率は 1 に近づいている．つまり，これは並列処理における計算粒度が大きくなる([16×16/1]->[256×256/1])に従い，通信処理の影響は小さくなるため，高性能実装の効果が薄らぐためである．

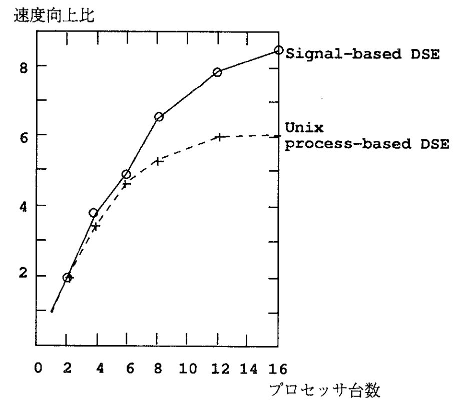


図 6 騎士巡回問題における速度向上比
Fig. 6 Speed-up ratio of Knight's Tour Problem.

（2） 騎士巡回問題

騎士巡回問題とは，N×N の盤面上のすべてのマスをチェスのナイト(騎士)が 1 回だけ通る経路を求める探索問題である．ここでは，Signal-based DSE と Unix process-based DSE における台数効果を調べる．

騎士巡回問題の並列処理において，プロセッサ数を変化させた場合の速度向上比を**図 6** に示す．Signal-based DSE, Unix process-based DSE 共に，プロセッサ台数が少ない場合には速度向上比は台数に比例して伸びている．しかし，台数が 8 台，10 台とさらに増えるにつれて，Unix process-based DSE では速度向上比が頭打ちになるのが認められる．一方，Signal-based DSE ではさらに台数が増加しても速度向上比が向上している．この結果から，Signal-based DSE では従来よりも多くのプロセッサを使用して高い並列度の問題解法を利用できることが分かり，高性能実装法の有効性が確認できた．

（3） マンデルブロー集合の描画計算

マンデルブロー集合とは，式(1)で定義される複素関数によって写像を無限に繰り返しても，z が無限にならないような C の集合のことである．

$$fc(z)=z*z+C \quad (1).$$

マンデルブロー集合から遠く離れている C に対して，この写像を繰り返すと，数回の写像で z は発散する．一方，集合に近い C に対しては，発散するまでの写像回数が次第に多くなる．この集合をグラフィックスとして描画するために C を平面座標とし，その C に対

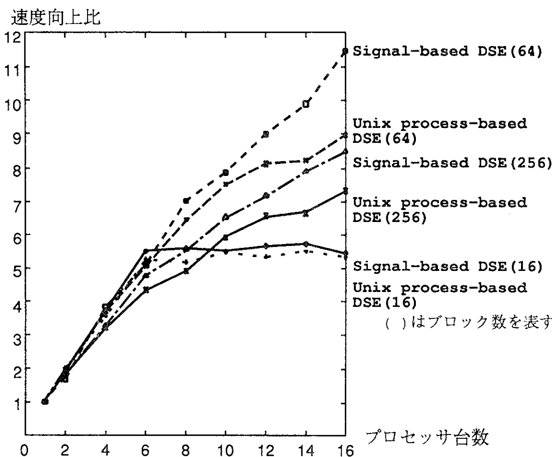


図 7　マンデルブロー集合の描画における速度向上比
Fig. 7　Speedup ratio of Mandelbrot Set.

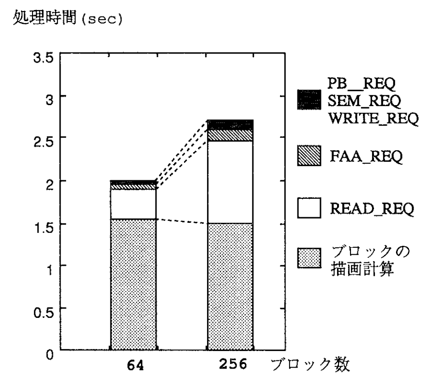


図 8　マンデルブロー集合の描画における処理時間の内訳
Fig. 8　Execution time of Mandelbrot Set in detail.

して行われた写像回数を輝度として表現する．

実際の描画計算は，全体として 256×256 の計算領域を複数のブロックに細分化し，ブロックごとに並列計算を行った．この計算過程には依存関係がないので，ブロックへのプロセッサ割り当ては，Fetch & Add プリミティブを用いてプロセッサが処理すべきブロックを実行時に獲得する動的負荷分散方式を採用している．ここでは，ブロック数によって計算の粒度を変えながら，プロセッサ数による台数効果を調べた．並列処理による速度向上比を**図 7** に示す．速度向上比は，ブロック数に関わらず，Signal-based DSE の方が Unix process-based DSE よりも改善していることが分かる．

ブロック数が 16 と少ない場合は，プロセッサ数を 6 台以上にしても速度向上比は頭打ちになっている．マンデルブロー集合の描画計算では領域によって計算量が均一でないため，ブロック数が少ないとブロックごとの計算量の差が大きくなると共に動的負荷分散の効果も現れにくく，プロセッサ間の計算時間にばらつきが生じることとなる．その結果，最終的な完了同期待ちにより速度向上比が頭打ちになっていると言える．一方，ブロック数が多い場合，すなわちブロック数が 64 と 256 の場合を比較すると，Signal-based DSE および Unix process-based DSE は共に，ブロック数が 64 の方が速度向上比が大きいという結果となった．これは，ブロック数が 256 と多くなると，ブロックごとの計算量の差が小さくなり，また動的負荷分散の効果も期待できる反面，ネットワークを介した共有メモリへのアクセス回数が多くなるためと考えられる．

そこで，ブロック数が 64 と 256 の場合について，

表 2　プリミティブ処理の詳細
Table 2　Processing time of the primitives in detail.

	ブロック数 64		ブロック数 256	
プリミティブ	処理時間(秒)	回数	処理時間(秒)	回数
PB_REQ	0.027388	15	0.055537	15
SEM_REQ	0.002532	18	0.011760	18
WRITE_REQ	0.005325	6	0.017799	26
FAA_REQ	0.062112	80	0.157078	272
READ_REQ	0.401887	64	1.044200	256

Signal-based DSE における内部処理の解析を行った．処理時間の内訳を**図 8** に示す．各ブロックの処理時間については，両者の間で大きな違いは見受けられない．処理時間全体の違いは，プリミティブの処理時間によって発生していることが明らかになった．よって，プリミティブ処理の内訳について詳細に調査を行った．その結果を**表 2** に示す．なお，表中の項目「処理時間(秒)」とは，項目「回数」分の合計時間を示している．このアプリケーションで用いられているプリミティブは，並列分岐を行うための PB_REQ，セマフォによる終了同期を行うための SEM_REQ，動的負荷分散に用いる FAA_REQ，そして共有メモリへのアクセスに用いる WRITE_REQ と READ_REQ である．ブロック数が 256 の場合は 64 の場合と比べて，ネットワークを介した共有メモリへのアクセスを行う READ_REQ や FAA_REQ のプリミティブ実行回数が約 4 倍に増加し，プロトコル処理などのネットワーク通信処理の影響が大きくなってしまうことが分かった．なお，ブロック数の増加（1 ブロックの領域の大きさの縮小）によって，READ_REQ プリミティブの実行 1 回あたりの処理時間は減少することに注意されたい．このことは，分散システムにおけるネットワークの通信処理が高速になれば，DSE の性能がさらに改善されることを

示唆している．今後，ATM 網に代表されるような超高速ネットワークを利用した分散システムが普及してくると，本稿で提示した高性能実装法の効果はハイパフォーマンスコンピューティング環境を実現する際にいっそう顕著になると期待できる．

5. おわりに

大学における最近の教育用計算機環境では多数のワークステーションが配置されており，並列処理環境としての潜在的可能性を備えているが，多数のユーザで共同利用している環境では OS 等のシステムソフトウェアを気軽に変更することは難しい．ハイパフォーマンスコンピューティング環境 DSE は，既存 OS に手を加えない方針の下に，分散システム上に共有メモリモデルに基づく並列処理環境を実現している．しかし，ユーザレベルでの実装を行うが故に，我々の初期の実装法では処理オーバヘッドが大きくなり，効率良い並列処理を妨げていた．

本稿では，通信処理における入出力関数の呼び出しに着目し，今まで別々の UNIX プロセスで実現していた DSE カーネルと DSE プロセスの処理を1つの UNIX プロセス内で行うことによる高速化の手法を提示した．さらに，DSE カーネルと DSE プロセスの処理を独立に扱う方法として，スレッドあるいはシグナルを用いる2種類の DSE の実装を試みた．その結果，ローカル通信におけるプリミティブ処理においては，通信処理の影響を完全に排除することができ，大幅な処理の高速化が図れた．また，リモート通信の処理については，スレッドを用いて実装した場合に処理の遅滞を引き起こすことが判明する一方，シグナルハンドラに DSE カーネルを実装した場合は入出力関数の呼出しの削減による処理速度の向上を確認できた．さらに，実際のアプリケーションを用いた性能評価を行うことにより，提示した実装法に基づく DSE が初期の DSE に比べて処理効率が向上していることを示し，さらに性能の向上を図るにはネットワークの通信処理の高速化が重要であることを明らかにした．

今後の課題としては，DSE における並列アプリケーションのプログラム開発を支援する環境の整備がある．現在，ユーザが DSE 用の並列アプリケーションを開発する際に，プログラミングからデバッグまでを支援する DSE 開発支援環境（DDD）について研究を進めており，より使い易い並列処理環境を目指す予定である．また，ATM などの超高速 LAN の利用を前提として，ユーザの使い易さを意識したメモリ R/W 方式の共有メモリを DSE に実装することも検討している．

謝辞　本研究を遂行するにあたり，実験データ収集に御協力いただいた本学情報工学研究科院生の岡　雅樹氏，ならびに日頃ご討論いただく有田・末吉研究室の諸氏に感謝の意を表す．なお，本研究の一部は文部省科学研究費（重点領域研究（1）課題番号04235103「超並列ハードウェア・アーキテクチャの研究」）の補助を受けたことを付記する．

参 考 文 献

1) Cheriton, D. R.: The V Distributed System, *Comm. ACM*, Vol. 31, No. 6, pp. 314-333 (1988).
2) Rozier, M., Abrossimov, V., Armand, F., Boule, I., Gien, M., Guilemont, M., Hermann, F., Kaiser, C., Langlois, S., Léonald, P. and Neuhauser, W.: Chorus Distributed Operating System, *Computing Systems*, Vol. 1, No. 4, pp. 305-370 (1988).
3) Sunderam, V. S.: PVM: A Framework for Parallel Distributed Computing, *Concurrency: Practice and Experience*, Vol. 2, No. 4, pp. 315-339 (1990).
4) 藏前健治，中條拓伯，前川禎男：ソフトウェア DSM におけるコヒーレント・キャッシュシステムの実装と評価，「並列処理シンポジウム JSPP'94」論文集，pp. 303-310 (1994).
5) Tezuka, T., Ryoukai, K., Apduhan, B. O. and Sueyoshi, T.: Implementation and Evaluation of Distributed Supercomputing Environment on a Cluster of Workstations, *Proc. 1992 Int. Conf. Parallel and Distributed Systems*, pp. 58-65, Taiwan, R. O. C. (1992).
6) Apduhan, B. O., Sueyoshi, T., Tezuka, T. and Arita, I.: The Effect of Communication Processing in Network Supercomputing Environment, *Proc. 7th Int. Joint Workshop on Computer Communications*, pp. 373-380 (1992).
7) Apduhan, B. O., Sueyoshi, T., Namiuchi, Y., Tezuka, T., Fujiki, T. and Arita, I.: Experiments and Analysis toward Distributed Supercomputing on a Distributed Workstation Environment, *Proc. 1991 Int. Symp. Supercomputing*, Japan, pp. 182-190 (1991); or *Supercomputer*, Special Issue for ISS '91 (Revised Version), Vol. 8, No. 6, pp. 90-100, Strichting Academisch Rekencentrum Amsterdam (SARA) (1991).
8) 手塚忠則，了戒　清，末吉敏則：分散システムを利用した並列処理環境における通信処理の影響，情報処理「マルチメディア通信と分散処理」ワークショップ論文集，pp. 249-256 (1993).
9) Wilson Ho, W., Olsson, R. A.: An Approach to Genuine Dynamic Linking, *Software-Practice and Experience*, Vol. 21, No. 4, pp. 370-390

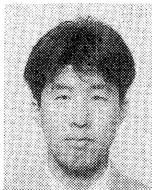
(1991).

10) *SUN Programming Utilities and Libraries*, pp. 1-47, SUN Microsystems (1990).

(平成6年9月16日受付)
(平成7年2月10日採録)

大西　淑雅（正会員）

1966年生．1989年九州工業大学工学部情報工学科卒業．同年九州工業大学情報科学センター助手．分散システム，計算機ネットワークの研究に従事．電子情報通信学会会員．



了戒　　清（正会員）

1970年生．1992年九州工業大学情報工学部知能情報工学科卒業．1994年九州工業大学大学院情報工学研究科情報科学専攻修士課程修了．同年富士通株式会社入社．分散システムの研究およびトランザクション処理に関する開発に従事．

末吉　敏則（正会員）

1953年生．1976年九州大学工学部情報工学科卒業．1978年同大学院工学研究科情報工学専攻修士課程修了．同年九州大学工学部情報工学科助手．同大学院総合理工学研究科助教授を経て，1989年より九州工業大学情報工学部知能情報工学科助教授．工学博士．計算機アーキテクチャ，システムソフトウェア，計算機ネットワーク，LSI設計などの研究に従事．著書「並列処理マシン（共著，オーム社）」．