Vol. 30 No. 12 情報処理学会論文誌 Dec. 1989

ネットワーク仮想記憶システムにおける宣言的アクセス制御方式†

陣崎 明†† 樋口昌宏††

我々は分散型密結合システムの構築を目的とし，ネットワーク結合による共有メモリ型マルチプロセッサ構成方式であるネットワーク仮想記憶システム (NET-VMS: NETworked Virtual Memory System) 方式，および NET-VMS 方式におけるプロセス間通信方式として宣言的プロセス間通信方式を提案している．本論文では NET-VMS 試作ハードウェア上で分散 OS を試作し，宣言的プロセス間通信方式の基礎となる宣言的アクセス制御方式による並列マージソートプログラムの処理実験を行った結果について述べる．ネットワーク結合で共有メモリを実現する場合，メモリデータの高速な移動と，共有メモリ管理制御の高速化が重要となる．NET-VMS は各プロセッサの仮想記憶システムを 100 Mb/s トークンリングネットワークで直接結合し，ハードウェア制御により最大 12 MB/s の実効通信性能を実現するとともに，仮想記憶ページごとに設けたメモリ管理用のハードウェアタグを用いて共有メモリ管理の高速化を実現している．宣言的アクセス制御方式は NET-VMS のハードウェアタグを用いることにより，セマフォを用いないプロセス間通信（排他制御，同期制御）を実現する．これらの方式に基づき最大 5 プロセッサの NET-VMS 上でマージソートプログラムのサブルーチンを粒度とする並列処理実験を行い，60% 以上の並列処理の台数効果を確認した．実験により提案方式による並列処理の実現性，ならびに宣言的アクセス制御による並列処理プログラミングの容易性を確認できた．

1. はじめに

マルチプロセッサシステムの実用化が進むにつれ，システム構成もバス結合を中心とした小規模な密結合システムから，これら小規模システムをさらにネットワークで結合したより大規模な分散システムへと拡大しつつある．これに伴い，並列処理における最も大きな問題であるプロセス間通信を，ネットワーク環境で高速化することが重要な課題となってきた．この問題解決のために，我々はネットワーク結合型マルチプロセッサシステム構成方式としてネットワーク仮想記憶システム (NET-VMS: NETworked Virtual Memory System) 方式[1]を，NET-VMS 方式におけるプロセス間通信方式として宣言的プロセス間通信方式[2] を提案している．これらの提案に基づき，今回，NET-VMS 試作システム上に並列マージソートプログラムを実装し，性能評価を行った結果，宣言的プロセス間通信方式の基礎となっている宣言的アクセス制御方式およびネットワーク結合型マルチプロセッサにおける並列処理の実現性を確認することができた．

本論文では，まず第 2 章でネットワーク結合型マルチプロセッサシステムを構築する上で解決すべき課題について述べる．次に第 3 章で宣言的アクセス制御方式の基本的な考え方を説明する．第 4 章では宣言的アクセス制御方式の実現例として，並列マージソートプログラムについて述べる．第 5 章では実験システムの概要，特に試作した分散 OS プロトタイプ DAX-I の構成について述べる．第 6 章では実験結果を示し，宣言的アクセス制御方式の有効性について考察する．

2. 従来技術と課題

ネットワーク結合による分散型コンピュータシステムのプロセス間通信方式としてはメッセージ通信による方法と共有メモリによる方法がある．メッセージ通信に基づくシステムの代表的な例としては TANDEM 社の GUARDIAN[3] やカーネギーメロン大学の Mach[4] 等がある．メッセージ通信はパケット通信で自然に実現可能であり，ネットワーク結合システムに適した方法であるが，反面，メッセージ（パケット）を組み立て，解読するメッセージ処理オーバヘッドや Ethernet の TCP/IP に代表されるネットワークレベルの通信制御オーバヘッドが大きいという問題がある．これらのオーバヘッドが通信速度を制限しており，仮に伝送路速度 100 Mbps 以上の高速ネットワークを使用しても，それが直ちにコンピュータ間通信の実効通信速度（実データの通信速度）向上に反映されないという結果になっている．

通信制御オーバヘッドを軽減する試みとしてはカリ

† Declarative Access Control Scheme on Networked Virtual Memory System by AKIRA JINZAKI and MASAHIRO HIGUCHI (Computer Networks & Visual Communication Lab., Fujitsu Laboratories Ltd.).

†† (株)富士通研究所情報通信研究部第一研究室

フォルニア大学バークレイ校の Sprite[5] や Bell 研究所の Meglos[6] 等がある．いずれも通信制御の簡略化による実効通信速度の向上を目指しているが，Sprite で 700 KB/秒（伝送路速度 10 Mbps, 4 KB転送）と伝送路速度に対して最高 57% 程度の実効通信速度しか得られていない．

一方，プロセス間通信を共有メモリで実現する方式ではメッセージ処理が不要となるため，処理オーバヘッドが削減できる利点がある．また，メッセージ通信では通信データが任意長であるのに対し，共有メモリ通信では固定長のメモリブロックとなるため通信制御の簡略化が容易となる利点もある．例えば Apollo 社の AEGIS[7] では伝送路速度 12 Mbps のネットワークを用いて 1000 KB/秒（1 KB 転送）と，伝送路速度の 85% の実効通信速度を実現している．ただし，共有メモリ方式では通信遅延の大きいネットワーク環境で共有メモリ制御（memory consistency 制御，メモリアクセスに対する排他制御，プロセッサ間のアクセス同期制御）を効率良く実現する点が大きな課題となる．また共有メモリシステムにおいて排他制御，同期制御はセマフォを用いて実現されるが，このため特定のメモリアドレスにアクセスが集中する Hot Spot 問題[8] が性能上の大きな課題となっている．

3. 提案方式

我々の提案方式の基本的な考え方は共有メモリ方式に基づく．分散したメモリをネットワーク結合し，ネットワークハードウェアによって通信制御および共有メモリ制御を実現することにより，ソフトウェアからハードウェアへ処理をオフロードすることによるプロセッサ負荷の軽減，ハードウェア処理による伝送路速度に比例した実効通信性能の実現，ネットワーク機能による共有メモリ制御の効率化を狙う．

3.1 ネットワーク仮想記憶方式

まず，宣言的アクセス制御方式の前提となっている NET-VMS 方式について説明する．

共有メモリをネットワーク結合によって実現するためには，通信遅延の大きい環境で共有メモリ制御とデータの移動を高速に実現することが問題となる．NET-VMS 方式は分散して配置したコンピュータ＝プロセッサノード (PN) の仮想記憶システムをブロードキャストネットワークで結合し，全体を一個の共有仮想記憶システムとして構成する（図 1）．各 PN はブロードキャスト通信機構と仮想記憶機構を組み合せたハードウェアを持つ．このハードウェアはネットワークを常に監視し，ブロードキャストパケットに格納された仮想記憶ページアドレスと処理要求に従って自 PN の仮想記憶制御テーブルを適切に変更することにより，仮想記憶ページ単位に共有メモリ制御を行う．この制御方法はマルチキャッシュシステムで用いられる snooping cache[9] に似ているが，ネットワーク経由で効率的な制御を実現するためのタグを仮想記憶ページごとに備えている点が異なる．タグには VALID, LOCK, COPY, SYNC, WAITER, PRIVATE がある（表 1）．共有メモリ制御のため，以下の二つのネットワーク処理を用いる．

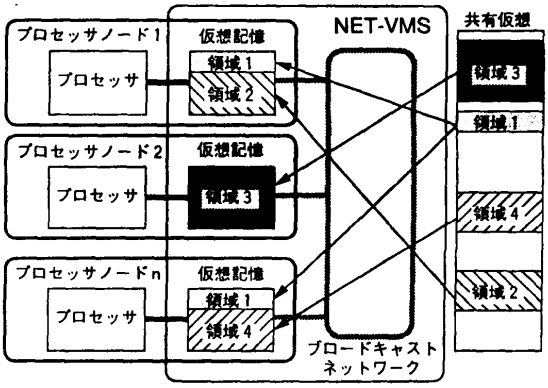

図 1 NET-VMS の構成
Fig. 1 Configuration of NET-VMS.

表 1 メモリタグ
Table 1 Memory tags.

VALID	自 PN 中にデータが存在することを示す
LOCK	データに排他アクセス中であることを示す
COPY	データが複数の PN に存在することを示す
SYNC	他 PN がアクセスするまで自プロセッサがアクセスできないことを示す
WAITER	排他アクセス中に他 PN からアクセス要求があったことを示す
PRIVATE	他 PN と共有しないローカルなページであることを示す

（1） Copy 処理：プロセッサが自 PN の実メモリに存在しない仮想記憶ページにアクセスした時，そのページを他 PN の実メモリから複写してくる．

（2） Unify 処理：プロセッサがライトアクセスしようとした時，他 PN の実メモリに存在する仮想記憶ページを無効化し，自 PN のページのみを有効とする．

NET-VMS 方式に基づいて伝送路速度 100 Mbps のネットワークを用いた試作システムを開発し，性能を実測した結果，4 KB の仮想記憶ページの Copy 処

理について 300～400 μs, Unify 処理について 30～50 μs を実現している[10]. 実効通信速度は 12 MB/秒 (4 KB 転送) で伝送路速度の 96% である.

3.2 宣言的アクセス制御方式

プログラム動作前に使用するメモリ領域へのアクセス条件 (例えばリードのみ, 排他アクセス等) を明示的に宣言しておき, プログラム動作中は NET-VMS のハードウェアによる共有メモリ管理機能を用いてメモリシステムが宣言されたアクセス条件を満たした上でプログラムにメモリアクセスさせる方式を宣言的アクセス制御方式と呼ぶ.

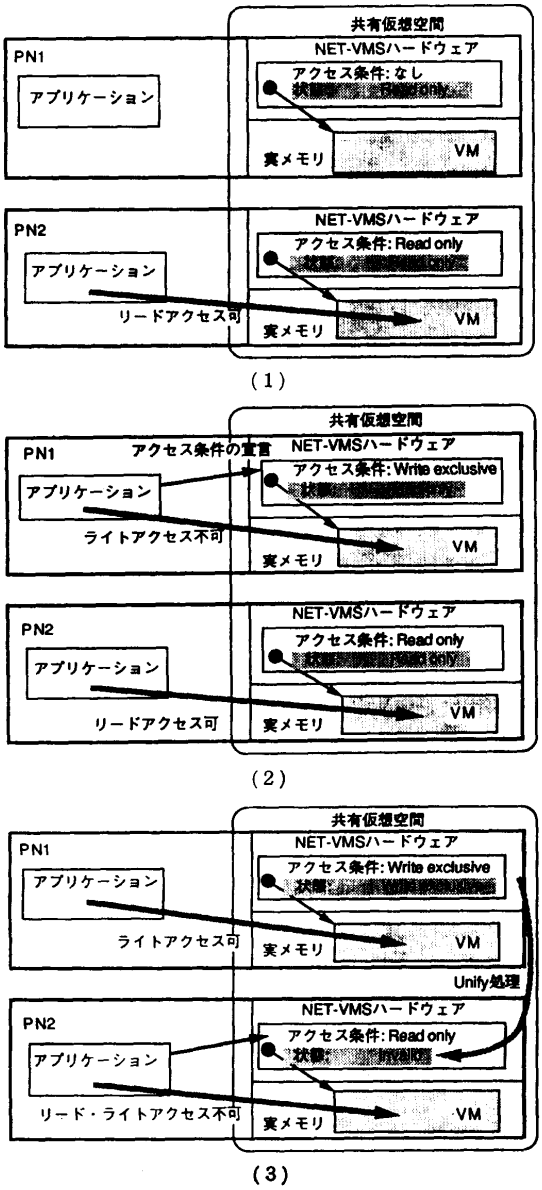

図 2 宣言的アクセス制御方式
Fig. 2 Declarative access control scheme.

NET-VMS における宣言的アクセス制御方式の処理を例を用いて説明する. あるプロセッサノード (PN 1) で動作しているプログラム (P 1) が共有仮想記憶空間上のある領域 VM を排他的ライトアクセスする場合を考える (図 2).

(1) 初期状態として他のプロセッサノード (PN 2) のプログラム (P 2) が同じ領域 VM をリード (Read only) 中で, 領域 VM が PN 1, PN 2 のメモリに Read only 状態で存在しているとする. (図 2 (1))

(2) P 1 は処理開始前に仮想記憶システムに対して領域 VM へのアクセス条件として Write exclusive を宣言する. (図 2 (2))

(3) P 1 は処理を開始し, 領域 VM をアクセスするが, この時領域 VM は Read only 状態であり, 先に宣言した条件と異なるため, アクセスフォールトとなる.

(4) P 1 の仮想記憶システムはこのアクセスフォールト要因を解析し, 領域 VM を Write exclusive 状態とするよう Unify 処理を行う. この結果 PN 2 が持つ領域 VM は Invalid, PN 1 が持つ領域 VM は Write exclusive となる. (図 2 (3))

(5) (4)の後, P 1 は(3)で中断した処理を継続することができる. この後, P 2 が領域 VM をアクセスしようとするとアクセスフォールトで停止し, 今度は PN 2 の仮想記憶システムが領域 VM を PN 1 から複写 (Copy) してくるまで待つことになる.

宣言的アクセス制御方式の時間的な流れを図 3 に示す. 本方式によれば, プログラムがアクセスする領域に対するアクセス条件が決まっていれば, プログラムの中でセマフォなどによる排他制御や同期制御を考慮することなく, 共有メモリ制御を実現できる点が重要

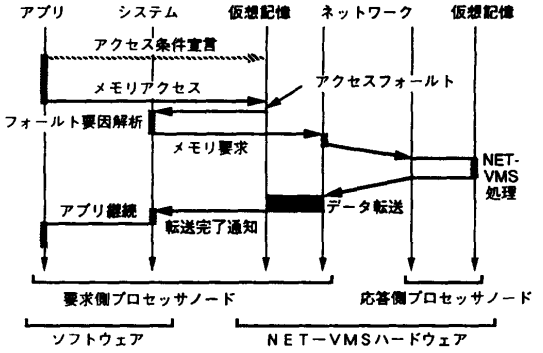

図 3 宣言的アクセス方式の時間的な流れ
Fig. 3 Timing chart of declarative access scheme.

である．言い換えるとアクセス条件を（プログラム動作前という意味で）静的に宣言しておけば，共有メモリ制御処理をプログラムの中に明示的に盛り込む必要がなくなる．並列処理プログラムでは処理するデータ(の格納された領域）の依存関係に従ってプログラム間の同期をとるのが一般的であるから，このデータの依存関係に基づいて宣言を行うことは困難ではない．

4. 並列マージソートプログラム

ここでは宣言的アクセス制御に基づく並列マージソートプログラムの構造，動作を説明することにより，宣言的アクセス制御方式の実現性を示す．

C言語で記述した並列マージソートプログラムを図4に示す．このプログラムはソートする配列 source を QSSIZE（この場合 2048 B）になるまで分割し，QSSIZE 以下になるとクイックソートを用いてソートする．次にこのソートした配列二つをマージしていき，最終的に結果を配列 result に格納して処理を終わる．

msort は配列 source へのポインタと配列の長さを引数とし，結果として新たに割り付けた配列 result へのポインタを返す．ms はマージソート本体で，source へのポインタ，source の長さ，result へのポインタを引数とし，source をソートした結果をresult に格納して返す．

```
#define QSSIZE 2048

long *
msort(source,length) long *source,length; {
long *result;
  result= valloc(length*sizeof(long));
   ms(source,result,length);                    ← 結果領域の割り付け
   return result;
}

void
ms(source,result,length)                        ← マージ結果を結果領域にかき込むという宣言
long *source,*result,length;
{
long halflen,*res1,*res2;
declare(result,length,Write exclusive create);
   halflen= length/2;
   if (halflen<=QSSIZE) {
      qsort(source,halflen);
      qsort(source+halflen,halflen);
      merge(source,source+halflen,result);
   } else {
      res1= valloc(halflen*sizeof(long));
      res2= valloc(halflen*sizeof(long));
      vfork(ms,source,res1,halflen);            ← 並列処理プロセスの生成
      ms(source+halflen,res2,halflen);
      merge(res1,res2,result);
      vfree(res1); vfree(res2);
   }
release(result,length);                         ← 結果領域の解放
}
```

A

図 4 並列マージソートプログラム
Fig. 4 Parallel merge sort program.

4.1 ms の処理

ms は source のデータ（この例では 4 B の数値データ）をソートした結果を result に格納して呼び出し元に復帰する．

（1） ms はまず与えられた source の長さ length を調べ，length/2 が QSSIZE より大きければ source を前半と後半に分け，前半をソートする ms を実行するプロセスを起動 (vfork) した後，後半をソートする ms を再帰的に自プロセスで実行する．この時，前半と後半をソートした結果を格納する配列 res 1, res 2 を新たに割り付け，res 1 を子プロセスに渡す．ここで res 1, res 2 は仮想記憶上の領域のみ割り付けられ，実メモリは割り付けられていない．

（2） (1)で再帰的に実行した ms が終了した後，res 1 と res 2 の配列のマージに移る (merge) が，マージのために配列 res 1 をリードアクセスした時点でアクセスフォールトとなり，子プロセスが結果を res 1 に書き込み終わるまで待たされることになる．ここでプロセス間同期が暗黙のうちに実現されていることになる．

（3） length/2 が QSSIZE 以下の時，ms はクイックソート (qsort) により source の前半と後半を source 上でソートし，結果を result へマージする．この場合 source のソートは並列実行されないことになる．

4.2 データ領域の管理

ms が使用するデータ領域は最初にトップレベルの ms に引数で与えられる source, result, 処理の過程で割り付けて使用される res 1, res 2（この数は source の分割数に比例する）である．

まず source についてはクイックソートされる長さになるまで分割され，それぞれ別の ms によってソートされた上でマージされる．このソートとマージは一つのプログラムが行うので source に関して並列プログラム間のデータ依存関係はない．

次に ms がマージしたデータを result に書き込む時，実メモリはまだ割り当てられていないのでアクセスフォールトとなるが，Write exclusive, create と宣言されている (declare) ので，この ms が動作している PN の仮想記憶システムが実メモリを割り付けて処理を続行させる．ここで create 条件は，この領域が仮想記憶のみ割り付けられており，ライトアクセス時に実メモリを割り付ける必要があることを表す．この機構によって ms がどの PN で実行されてもその PN の実メモリが割り付けられる．

result 領域については，この result にマージ結果を書き込むプロセス（子 ms）と，子 ms を起動したプロセス（親 ms）との間でデータ依存関係がある．親 ms は 4.1 節(2)で述べたように子 ms の result (親 ms からは res 1 や res 2) をリードしようとするが，子 ms が Write exclusive 条件でアクセスしているため他のプログラムはリードできない．子 ms がマージを終了してアクセス条件の解除を行った (release) 時点で仮想記憶システムは子 ms の result 領域を親 ms の res 1 や res 2 領域へ Copy し，親 ms がリード可能となる．以上に説明した並列マージソートの動作を**図 5** に示す．

4.3 セマフォによるプログラムとの比較

図 4 のプログラムとセマフォを用いた場合との記述性を比較する．図 4 の A の部分をセマフォ[11]を用いて記述したものを**図 6** に示す．図 6 のプログラムは以下の理由で複雑になっている．

・共有データのアクセス制御のために共有データと対応するセマフォ変数を導入しなければならない．

・P 命令（共有データへのアクセス開始時に行う．既にアクセス中のプロセスがあれば，そのプロセスが V 命令を実行するまで待つ），V 命令（共有データへのアクセス終了時に行う）をプログラム手続き中の適切な場所に埋め込む必要がある．

これに対し図 4 のプログラムでは，アクセス制御の記述が手続き記述の最初の declare と手続き記述の最後の release のみであり，特定の制御変数の導入や手続き中に制御命令を埋め込む必要がない．このように宣言的アクセス制御によりセマフォと比較して並列処理プログラム記述を簡明にすることができる．

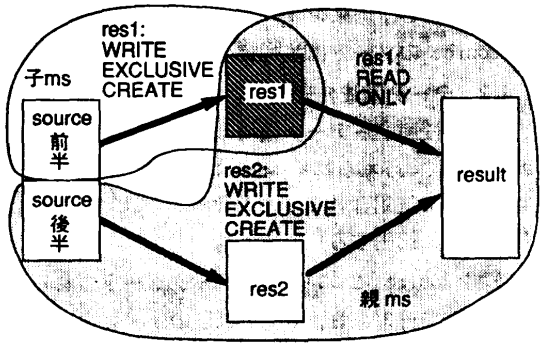

図 5 並列マージソートにおけるデータ依存関係
Fig. 5 Data dependency in parallel merge sort.

```
res1に対するセマフォ変数
 P(semres1);                          子プロセスのためのP命令
 process_generate(ms,source,
       res1,halflen,semres1);         子プロセスの生成
 ms(source+halflen,res2,halflen);
 P(semres1);                          マージのためのP命令
 merge(res1,res2,result);
 V(semres1);                          マージのためのV命令
 free(res1); free(res2);
}
V(semres);                            resultへのアクセス終了を
                                      表すV命令
resultに対するセマフォ変数
```

図 6 セマフォによる並列マージソートプログラム
Fig. 6 Parallel merge sort program using semaphore.

5. 実験システム

NET-VMS 試作システムの概要を**図 7** に示す．NET-VMS 試作システム上で宣言的アクセス制御方式に基づく分散オペレーティングシステムのプロトタイプ DAX-I を試作した．DAX-I は現在のところ，共有仮想記憶管理機能，プロセスディスパッチ機能しか持っていないが，先に説明した並列マージソート程度のアプリケーションを並列処理可能である．

DAX-I は例外処理部，プロセスディスパッチャ，宣言的アクセス制御特有の機能（declare 等）を処理するシステムコールルーチンからなる．例外処理部はメモリアクセスフォールト時のフォールト要因の解析

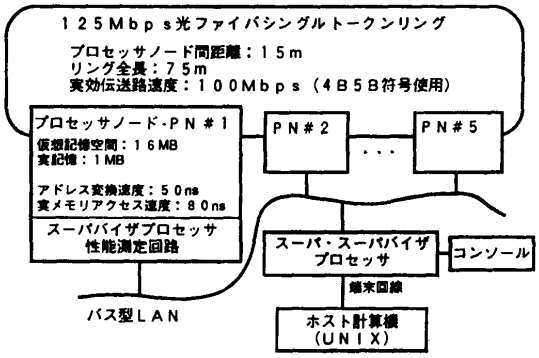

図 7 NET-VMS 試作システムの構成
Fig. 7 NET-VMS prototype system configuration.

表 2 DAX-I のモジュール
Table 2 DAX-I modules.

		ステップ数
sysinit	共有仮想記憶初期化	150
exception	例外処理ハンドラ	200
dispatch	プロセススケジューラ	150
vfork	グローバルプロセス生成	50
valloc	共有仮想記憶割り付け	50
vfree	共有仮想記憶解放	50
declare	アクセス条件宣言	30
release	アクセス条件解除	30

計 710 ステップ (2300 バイト) 68010 アセンブラ使用

と解析結果に応じた処理を行う部分で，共有仮想記憶空間のメモリ管理と NET-VMS 制御の機能を持つ．プロセスディスパッチャは共有仮想記憶上のプロセス管理テーブルを調べ，実行すべきプロセスを見つけて実行する．システムコールは declare, release, valloc, vfree, vfork の5種類である．DAX-I のモジュール構成とコードサイズを表2に示す．DAX-I は 68010 アセンブラで記述している．

DAX-I は共有仮想記憶管理テーブル，プロセス管理テーブルを共有仮想空間上に置き，単一カーネルイメージで動作する．すなわち，システム全体で一個の仮想記憶管理テーブルと一個のプロセス管理テーブルを持ち，これを各プロセッサノードの DAX-I カーネル（分散カーネルと呼ぶ）が共有して動作する（図8）．分散カーネル間の共有メモリアクセス制御も宣言的アクセス制御方式によっている．

共有仮想記憶管理テーブル (VMMT) は共有仮想記憶空間上の固定領域に置かれており，IPL 時，DAX-I が動作する前に初期化される．分散カーネルはアプリケーションからの valloc システムコールに対応して，VMMT を参照して仮想記憶の割り付けを行う．

DAX-I ではプロセス管理テーブルをグローバルプロセスプール (GPP) と呼ぶ．vfork システムコールがあると，分散カーネルは GPP 領域を取り込み，プロセスを書き込んでから GPP 領域を release する．一方，実行すべきプロセスを持っていない PN のプロセスディスパッチャは GPP をリードアクセスした状態でアクセスフォールトにより停止している．GPP が release されると NET-VMS がこれを自動的に停止している PN に転送し，プロセスディスパッチャを起動する．この結果，プロセス移動 (process migration) が行われる．

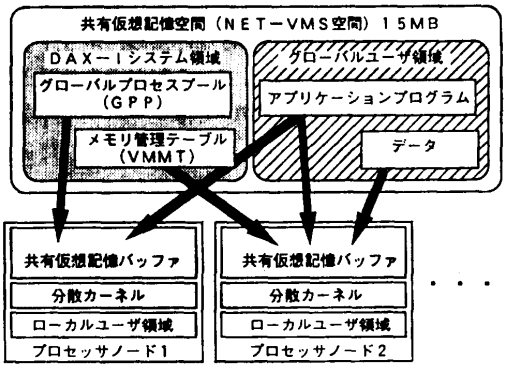

図 8 DAX-I の構造
Fig. 8 Structure of DAX-I.

6. 実験結果

実験システム上で並列マージソートを実行した結果を示す．ソートした配列は 4 B の数値データで，配列の大きさは 32768 個 (128 KB) である．

DAX-I の分散カーネルは各 PN に置き，一つの PN に並列マージソートプログラムと配列データを置いた状態からシステムを実行させた．したがって，他のプロセッサノードは並列マージソートプログラムと配列データをネットワークを介して獲得して処理を行う．仮想記憶ページ長は 4 KB, QSSIZE＝2 KB で実験した．

また，DAX-I, 並列マージソートプログラムとも，動作させる PN 数にかかわらず同じものを使用している．本システムは単一システムイメージで動作するので，ソフトウェアは PN 数を意識する必要が全くない．

表 3 システムコールのオーバヘッド
Table 3 System call overheads.

vfork	610 μs
valloc	30 μs
vfree	30 μs
declare	45 μs (4 KB)-450 μs (64 KB)
release	380 μs (4 KB)-5.6 ms (64 KB)

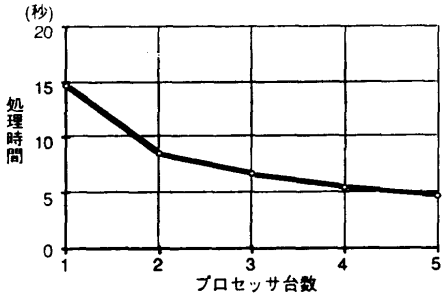

図 9 並列マージソート実行時間
Fig. 9 Elapsed time of parallel merge sort.

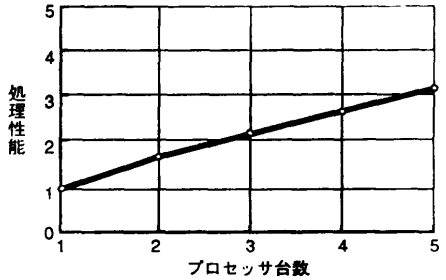

図 10 並列マージソートの台数効果
Fig. 10 Speedup in parallel merge sort.

まず DAX-I のシステムコールの処理時間を表 3 に示す．vfork 以外の処理は処理する仮想ページ数によって処理時間が変化する．

図 9 にマージソート処理時間とプロセッサ数の関係を図 10 に台数効果を示す．ほぼリニアに速度向上しているが，プロセッサ 1 台あたりの性能向上率は 60% である．ネットワークが処理ネックになっているとすれば PN 数が増加するにつれて性能向上率は低下するはずであるから，ネットワークの通信オーバヘッドよりも DAX-I 自体のオーバヘッドが支配的と考えられる．

図 11 に負荷分散の様子を示す．GPP を用いたスケジューリング方式によりほぼ均等な負荷分散が実現できていることがわかる．最も負荷の大きい PN はトップレベルのマージソートを実行した（すなわち最後のマージ処理を行った）PN である．

図 12 にマージソート処理時間の内訳を全プロセッサについて合計した結果を示す．システム処理時間にはアクセスフォールトハンドラ処理時間，プロセススケジューラ処理時間，アプリケーションが実行したシステムコールの処理時間，ネットワーク処理待ちの時間を含んでいる．アプリケーション処理時間は純粋にクイックソートおよびマージソートを行った時間である．アイドル時間は PN がプロセス待ちとなっている時間である．アイドル時間は PN 数の増加に従って増大しているが，最後のマージを一つの PN が処理している間，他の PN がアイドルになっている等マージソートアルゴリズムの並列度の限界によるものである．またシステム時間が 1～3 台で PN 数の増加に従って増大し，3～5 台では大きな差が表れていない．この場合，システム時間の大部分がネットワークを介したデータの転送に要するオーバヘッドであり，プロセス間でのデータの授受がネットワークを介して行われる比率は PN 数が 3 台以上でほぼ一定になっているためである．

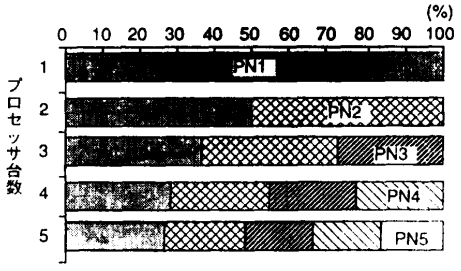

図 11 負荷分散
Fig. 11 Load distribution on parallel merge sort.

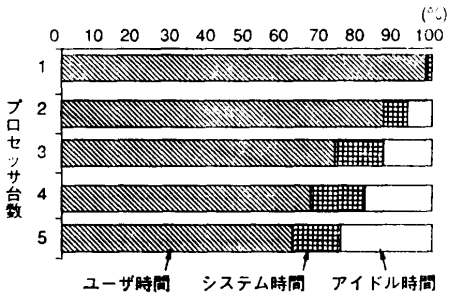

図 12 処理時間の内訳
Fig. 12 Processing time details.

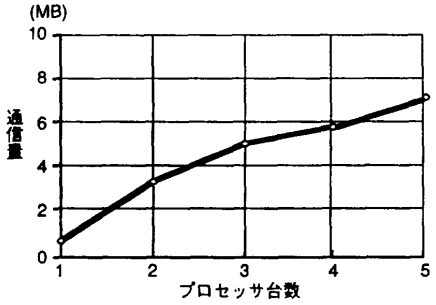

図 13 ネットワーク通信量
Fig. 13 Amount of network communication.

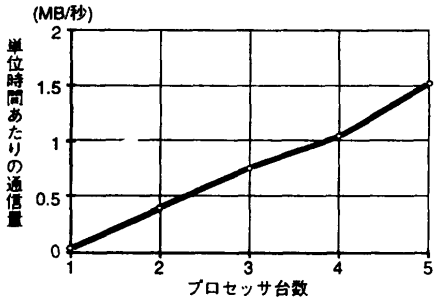

図 14 単位時間あたりの通信量
Fig. 14 Network communication rate.

図 13 は処理中のネットワーク通信量，図 14 は単位時間あたりのネットワーク通信量を示す．5 PN 動作時で約 7 MB の通信を行っている．実験システムでは瞬間最大通信量を測定できないため，ネットワーク通信量が処理性能を制限しているかどうか定量的に評価できないが，この点についてはマージソート処理中に他の独立な通信を行うことにより実験した結果，マージソートの処理時間には影響ないことがわかった．

以上の結果により次のことが確かめられた．

（1） 宣言的アクセス制御方式が実現でき，並列処理可能である．

（2） 共有メモリ制御を意識することなく，並列処理プログラムを簡明に記述できる．

(3) プロセッサ数に依存しない並列処理プログラムが実現できる.

(4) NET-VMS 方式を用いたネットワーク結合型マルチプロセッサシステムにおいて, サブルーチンレベルの粒度 (granularity) の並列処理が実現可能である.

7. おわりに

ネットワーク仮想記憶方式で実現可能な宣言的アクセス制御方式を提案し, この方式に基づく分散型オペレーティングシステムプロトタイプとして DAX-I を試作した. このシステム上で並列マージソートプログラムを実行し, 宣言的アクセス方式による並列処理プログラムの記述性, 実行可能性, および最大5ノードまでの分散環境下での有効性を検証した.

DAX-I は不必要なネットワーク処理を行っている等の問題があるためシステムオーバヘッドが大きい. この点を改善することにより, さらに良い結果が期待できる. また, 結果をより詳細に解析するためには各 PN の動作状況を時系列的に観測する必要があることがわかった. この点については今後の課題と考えている.

今後は DAX-I の改善と同時に多種の並列処理アプリケーションを実装, 評価し, 宣言的アクセス方式の適用性ならびに NET-VMS と宣言的アクセス制御による並列処理の粒度の限界を調べていく予定である.

謝辞 研究の当初より御討論, 御指導を頂いた(株)富士通研究所情報通信研究部八星部長, 本論文をまとめるにあたり有益な御助言, 御指導を頂いた同津田部長代理に感謝いたします. また, 試作システムの開発に御協力頂いた同第一研究室の玉野 肇氏, 栗田敏彦氏, ならびに加藤光幾氏に感謝いたします.

参考文献

1) 陣崎 明, 八星禮剛: ブロードキャストネットワークによる分散型単一階層仮想記憶システム, 信学技報, CPSY 86-20 (1986).
2) 陣崎 明, 八星禮剛, 樋口昌宏: ネットワーク仮想記憶システム NET-VMS (2)プロセス間通信方式, 第33回情報処理学会全国大会論文集, 3T-8 (1986).
3) 渡辺栄一編: フォールト・トレラント・システム, p. 414, マグロウヒルブック (1986).
4) Young, M., Tevanian, A., Rashid, R., Golub, D., Eppinger, J., Chew, J., Bolosky, W., Black, D. and Baron, R.: The Duality of Memory and Communication in the Implementation of a Multiprocessor Operating System, *Proc. of 11-th Symp. on Operating Systems Principles*, pp. 63-76 (1987).
5) Ousterhout, J., Cherenson, A., Douglis, F., Nelson, M. and Welch, B.: The Sprite Network Operating System, *Computer*, Vol. 21, No. 2, pp. 23-36 (1988).
6) Gaglianello, R.: Meglos: An Operating System for a Multiprocessor Environment, *Proc. of 5th Int. Conf. of Distributed Computing Systems*, pp. 35-42 (1985).
7) Nelson, L. and Leach, P.: The Evolution of the Apollo DOMAIN, *Proc. of Hawaii Int. Conf. on System Science 17th*, pp. 470-479 (1984).
8) Pfister, G. and Norton, V.: "Hot Spot" Contention and Combining in Multistage Interconnection Networks, *Int. Conf. of Parallel Processing*, pp. 764-771 (1985).
9) Archibald, J. and Bear, J.: Cache Coherence Protocols: Evaluation Using a Multiprocessor Simulation Model, *ACM Trans. Comput. Syst.*, Vol. 4, No. 4, pp. 273-298 (1986).
10) 陣崎 明, 樋口昌宏, 栗田敏彦: 100 Mb/秒トークンリングネットワークによる高速コンピュータ間通信システムの試作, 信学技報, IN 88-55 (1988).
11) 土居範久: 並列処理の諸問題, 同期問題, 情報処理, Vol. 27, No. 9, pp. 1022-1030 (1986).

(平成元年5月31日受付)
(平成元年9月12日採録)

陣崎 明 (正会員)

1954年生. 1978年広島大学工学部電気工学科卒業. 1980年同大学院修士課程修了. 同年(株)富士通研究所入社. 以来, LAN, 計算機間通信システム, 分散型計算機システムに関する研究に従事. 電子情報通信学会会員.

樋口 昌宏 (正会員)

1959年生. 1983年大阪大学基礎工学部情報工学科卒業. 1985年同大学院修士課程修了. 同年(株)富士通研究所入社. 以来, 計算機間通信システムに関する研究に従事.