オペレーティング・システム　56−12
(1992. 8. 19)

MP UNIXにおけるデバイスアクセス手法の比較

河内谷 清久仁　　白鳥 敏幸　　山崎 秘砂
日本アイ・ビー・エム株式会社　東京基礎研究所

マルチプロセッサ・システムを設計する際に問題になる点の一つは,「どのプロセッサがデバイスアクセスを行なうか」ということである．特定のプロセッサのみがデバイスアクセスを行なう「非対称」な方式は，ハードウェア, OS ともに実現が容易である反面，任意のプロセッサからのデバイスアクセスを可能とした「対称」な方式に比べて性能が悪くなる可能性がある．この問題に関して，我々は実機上にいくつかのデバイスアクセス手法をインプリメントし，実際にどの程度の性能差がみられるか測定を行なった．本稿は，その結果及びそれに関する考察についてまとめたものである．なお今回の測定には，当研究所で試作したTCMP マシン TOP-1 に，対称型マルチプロセッシングをサポートする UNIX である OSF/1 を移植したものを用いている．

Comparison of device access methods in MP UNIX

Kiyokuni KAWACHIYA　Toshiyuki SHIRATORI　Hisa YAMASAKI

IBM Research, Tokyo Research Laboratory
5-19, Sanbancho, Chiyoda-ku, Tokyo, 102 Japan

One major problem in designing a multiprocessor system is how to access devices from multiple processors. The device access methods of MP systems can be categorized into two types — *asymmetric* methods and *symmetric* methods. Asymmetric methods allow only one specific processor to access devices, whereas symmetric methods allow all processors to access devices. The former are easier to implement, but may degrade the system performance. To compare these two types of method, we have implemented them on a real MP system, and evaluated their performance. This paper reports the results and some considerations on device access in MP systems. OSF/1 TOP-1 is used for the evaluation.

1 はじめに

最近，対称型マルチプロセッシング[1]をサポートしたUNIX**オペレーティング・システムがいくつも利用可能になってきている．これらの対称型MP UNIXでは，カーネル処理を複数のプロセッサ上で並列に実行することが可能になっており，非対称型MP UNIXにみられた「カーネル・ボトルネック[1, 2]」が，かなり解消されている．

我々は，MPオペレーティング・システム研究のテストベッドとするために，対称型MP UNIXの一つであるOSF/1**を，当研究所で開発したMPマシンTOP-1に移植した．この移植の際問題になった点の一つが，デバイスアクセスの方式である．デバイスアクセスはマシン依存の部分であり，移植の際には特に注意が必要となる．特に，カーネルが並列化されている場合，デバイスアクセスの効率がシステム全体の性能に，より大きな影響を及ぼすことが考えられる．例えば，デバイスへのアクセスをシリアライズしてしまうと，その部分がボトルネックになってしまい，カーネルを並列化した意味がなくなってしまうおそれもある．また，どのようなデバイスアクセス方式が可能かはハードウェアのアーキテクチャによっても左右される．

これらのことから，今後のマルチプロセッサ・システム設計のためにも，デバイスアクセスの方式がシステム全体の性能にどの程度影響を及ぼすか把握することは非常に有用なことと思われる．そこで我々は今回の移植にあたり，いくつかのデバイスアクセス方式をインプリメントし，実際にどの程度性能に差が出るかを測定してみた．本稿は，その結果及びそれに関する考察についてまとめたものである．まず，次章では，MP UNIXでのデバイスアクセスで考慮すべき2つの点を説明する．次に，評価環境として用いたOSF/1 TOP-1の構成を簡単に説明する．最後に，測定内容と測定結果を示し，それをもとに検討を行なう．

2 MP UNIXでのデバイスアクセス

マルチプロセッサ環境でのデバイスアクセスで考慮しなければならない問題点は，一言でいうと「どのプロセッサがデバイスアクセスを行なうか」という点である．(もちろん，複数プロセッサからの要求に対応するためには，ディスクアレイ[3]などによる「デバイス自身の性能アップ」も重要であるが，それについては本稿ではふれない．)UNIX系OSの場合，この問題は以下の2つに大きく分けられる．

問題1. どのプロセッサがデバイスに指令を出すか
問題2. どのプロセッサが割り込みを受けるか

この2つはそれぞれ，デバイスドライバの上半分，下半分(割り込みハンドラ)をどのプロセッサで実行するかということと対応している．

まず，**問題1**について考える．これには以下の2種類の方式がある．

1a. 特定のプロセッサのみがデバイスに指令を出す
…非対称な方式
1b. 任意のプロセッサがデバイスに指令を出す
…対称な方式

1aの非対称な方式では，デバイスにアクセスしようとしたプロセス(もしくはスレッド)が，指令を出せないプロセッサ上で実行されていた場合，指令を出せるプロセッサ上に移動してからデバイスアクセスをすることになる．この方式では，デバイスドライバの処理は特定のプロセッサ上にシリアライズされるので，従来のユニプロセッサ用UNIXのデバイスドライバを，並列化することなくそのまま利用できる．また，ハードウェア上の制約から特定のプロセッサしかデバイスにアクセスできない場合などにも有効である．次章で説明するが，OSF/1ではこの方式をサポートするためにファネリングという枠組を用意している．

一方，**1b**の方式では，各プロセッサがデバイスに対して指令を出すことになる．そのため，ロックプリミティブなどを利用して排他制御を行なうように，デバイスドライバを並列化する必要がある．

次に，**問題2**の割り込みの配送について考える．これにも大きく分けて2通りの方式がある．

2a. 常に特定のプロセッサに割り込む …非対称な方式
2b. 任意の適当なプロセッサに割り込む …対称な方式

2aの非対称な方式では，デバイスからの割り込みは全てあらかじめ決められた特定のプロセッサに送られる．一方，**2b**の方式では，割り込みは適当なプロセッサにダイナミックに配送される．割り込まれるプロセッサの選び方としては，そのI/Oを起動したプロセッサを選ぶ手法や，暇なプロセッサを選ぶ手法などが考えられる．

直観的には，問題1, 2ともにbの対称な方式を用いた方が性能が向上すると思われる．しかし実際の動作環境においては，デバイスの種類や，同時に動いているプロセスの数，I/Oアクセスの頻度などによって効果の程度が違ってくる可能性がある．また，bの方式は，ハードウェ

[1]本稿では，カーネル処理が特定のプロセッサで行なわれるものを「非対称型」，基本的にどのプロセッサでもカーネル処理が可能なものを「対称型」としている．

**UNIXはUNIX System Laboratories, Inc.の開発・許諾製品．
**OSF/1はOpen Software Foundation, Inc.の商標．

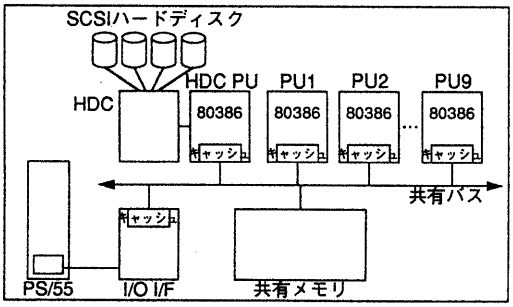


図 1: TOP-1 のシステム構成

アでの対称アクセスサポートや，デバイスドライバの並列化などの手間がかかるため，性能差によってはaの方式を選択する方が良い場合も考えられる．そこで我々は，ここで挙げた各手法について，OSF/1 TOP-1 上にインプリメントし，各種の条件下において実際にどの程度の差がみられるか測定を行なってみた．

3 評価システムの構成

評価に入る前にまず，本章では評価環境として用いたOSF/1 TOP-1 の構成を簡単に説明する．

3.1 TOP-1

図 1は TOP-1 のシステム構成を示したものである．TOP-1 は，10 台のプロセッシング・ユニット (PU) をスヌープ・キャッシュを介して共有バスで結合した，いわゆる密結合共有メモリ型のマルチプロセッサ (TCMP) ・マシンである [4, 5]．各 PU には Intel 80386 と 128K バイトのスヌープ・キャッシュが装備されている．キャッシュ間の一貫性はハードウェアによって保たれる．PU 間の同期機構としては，メッセージ割り込みの機能が用意されている．

TOP-1 から制御できる I/O デバイスとして 4 台の SCSI ハードディスクが用意されているが，このハードディスクに直接アクセスできるのは，ハードディスク制御 (HDC) カードに接続された特定の PU(図の HDC PU) のみである．その他の I/O や，外界へのアクセスは，フロントエンド・プロセッサとして接続されている IBM PS/55* を介して行なわれる．この PS/55 から，TOP-1 上の共有メモリにアクセスしたり，PS/55 と各 PU の間でメッセージ割り込みをかけることも可能である．

*PS/55, PS/2, 及び AIX は IBM Corp. の商標.

3.2 OSF/1

TOP-1 上では従来，AIX PS/2* をマスタ・スレーブ型で MP 対応化した TOP-1 OS [1] が動作していたが，我々は今回新たに OSF/1 の移植を行なった．OSF/1 はオープン・ソフトウェア財団 (OSF) が提供している UNIX オペレーティング・システムで，その特長の一つが，対称型マルチプロセッシングのサポートである [6]．

OSF/1 カーネルのベースとなっているのは，カーネギ・メロン大学で開発された Mach 2.5 [7] である．オリジナルの Mach 2.5 では UNIX サーバ (BSD サーバ) の部分はあまり並列化されておらず，unix_master と呼ばれる特定のプロセッサ上にシリアライズされて実行されていたが，OSF/1 ではこの部分もかなり書き直されており，ファイルシステムやネットワークなどの主要なモジュールは，どのプロセッサでも実行可能となっている．デバイスアクセスに関しても，基本的にどのプロセッサでもデバイスドライバのコードを実行できるようになっている．ただし，デバイスドライバ自身を並列に実行するためには，OSF/1 が提供するロックプリミティブなどを用いて並列化する必要がある．

従来の，並列化されていないデバイスドライバを利用するための枠組として，OSF/1 では「ファネリング (funnelling) [2]」という機構を用意している [8, 9]．ファネリングの機構を使う場合，デバイスドライバの登録時に「並列化されていない」というフラグを立てておく．OSF/1 カーネルは，デバイスドライバに入る直前にこのフラグをチェックする．もしフラグが立っていれば，そのプロセス (スレッド) を強制的に特定のプロセッサ (unix_master) に移動させる．これによって，そのデバイスドライバは unix_master プロセッサ上で，シリアルに実行されることになるため，従来のユニプロセッサ用 UNIX のコードをそのまま用いることができる．

3.3 OSF/1 TOP-1

OSF/1 の TOP-1 への移植は，OSF から提供されている AT386 用のコードを元に行なった．OSF/1 の機種独立な部分は既に MP 対応になっているため，一部のバグフィックスなどを除いてほぼそのまま使用している．機種依存部に関しては，AT386 用のものを TOP-1 OS のコードなどを元に MP 対応に書き直している．

移植の際の問題点の一つはデバイスアクセスである．上で述べた通り，TOP-1 ハードウェアでは，ハードディスクにアクセスできるのは HDC PU のみで，他の PU からは直接のアクセスができない．そこで今回の移植では，HDC PU 上でデバイス制御のための専用プログラムを走

[2] ファネル (funnel) とは「じょうご」の意.

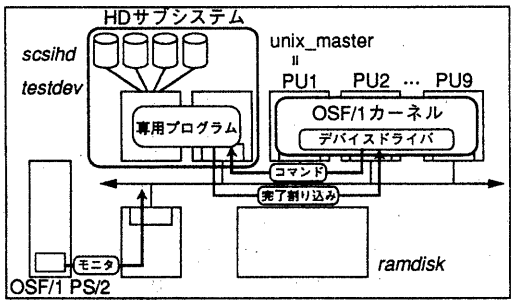


図 2: OSF/1 TOP-1 の構成

らせ,「プログラマブルな HD サブシステム」とみなせるようにしている. OSF/1 カーネルは残りの 9 台のプロセッサで実行される. これらのプロセッサからは, HD サブシステムに対して仮想的なハードディスクアクセス命令を出すことになる. HD サブシステムでは各プロセッサから送られたコマンドを処理して実際のハードディスクアクセスを行なう. 処理完了の通知は, HDC PU から適当な PU に対してメッセージ割り込みをかけることで行なわれる. このような構成をとることにより, OSF/1 TOP-1 では 2章であげたデバイスアクセスに関する問題点について, 様々な方式を試すことが可能になっている.

図 2が, 今回の性能評価に用いた OSF/1 TOP-1 の構成を示したものである. HD サブシステムにアクセスするためのデバイスドライバは並列化されており, 9 台の PU は対称に HD サブシステムにアクセスすることができる. また, 比較のために, PU1(unix_master) にファネルしてからアクセスするようにも構成できる. HD サブシステムはプログラマブルなので, 処理完了割り込みの配送先も各種のパターンを実現することができる. また, フロントエンドの PS/55 上では, PS/2 用に移植された OSF/1 が動作しており, ここから TOP-1 の共有メモリにアクセスすることで, OSF/1 TOP-1 の動作状況をモニタできるようになっている.

今回の性能評価用には, 3 種類のデバイスを用意した. 1 番目は, HD サブシステム上の SCSI ハードディスクである. 2 番目は, 今回のテスト用に用意したダミー・デバイスである. このデバイスは, HD サブシステム上で実現されており, アクセスがくると一定時間 (約 17msec) 待ったあとで処理完了割り込みを返すようになっている. 実際のハードディスクを用いたテストでは, ディスクのシークタイムのばらつきなどの影響が大き過ぎて性能の比較がうまくできない場合も考えられるため, このデバイスを用意した. 3 番目は, 共有メモリの一部をデータ領域に用いた RAM ディスクである. これらのデバイスドライバについても, ファネリングするかどうかを選択できるようにしてある. 以後の説明では, この 3 つのデバイスをそれぞれ "scsihd", "testdev", "ramdisk" と呼ぶことにする.

4 評価方法

いよいよ本稿の目的である, MP UNIX でのデバイスアクセス手法の比較に入る. 本章では, 比較したデバイスアクセス方式の説明と, その性能評価に用いた方法を説明する. 2章で述べた通り, テストを行なったのは次の点である.

問題 1. どのプロセッサがデバイスに指令を出すか
問題 2. どのプロセッサが割り込みを受けるか

この 2 点について, いくつかの方式を OSF/1 TOP-1 上にインプリメントし, 実際にどの程度の性能差がみられるか測定した.

問題 1 については次の 2 種類の方式をインプリメントして比較した.

1a. 特定のプロセッサのみがデバイスに指令を出す
1b. 各プロセッサがデバイスに指令を出す

1a の実現には OSF/1 のファネリング機構を用いている. デバイスにアクセスしようとしたプロセス (スレッド) は, まず PU1(unix_master) 上に移され, シリアライズされる. これに対し **1b** の方式では PU1 ～ PU9 の各プロセッサが直接デバイスアクセスを行なう. そのため, デバイスドライバ内でロックを用いた排他制御を行なっている. ファネリング機構を利用すると, デバイスドライバのインプリメントは容易になるが, 余計なプロセッサ移動が必要であることなどから, 1b の対称なアクセスに比べて性能は悪くなると考えられる. ただし, 1b の方式でもロックの用い方などによっては性能が低下する可能性もある.

問題 2 の割り込みの配送については, 以下の 6 種類の方式をインプリメントして比較を行なった.

2a. 常に特定のプロセッサに割り込む
  2a-1. 常に PU1(unix_master) に割り込む
  2a-2. 常に PU2 に割り込む
2b. 任意の適当なプロセッサに割り込む
  2b-1. その I/O を起動したプロセッサに割り込む
  2b-2. 「暇な」プロセッサに割り込む
  2b-3. 「忙しい」プロセッサに割り込む
  2b-4. 全プロセッサ (PU1 ～ PU9) に割り込む

ハードウェア上, 特定のプロセッサにしか割り込めないようなアーキテクチャでは, **2a** のどちらかの方式をとることになる. **2a-1** は, 常に unix_master に割り込む方式で, もっとも単純な方式である. 並列化されていないデ

バイスドライバをファネリング機構を用いて利用している場合は，この方式をとる必要がある．一方，**2a-2** は，unix_master 以外の特定プロセッサに割り込むというもので，unix_master が過負荷になっているような場合は 2a-1 よりも効果があるかもしれない．

2b の方式は，状況に応じて割り込み先を変えるというものである．**2b-1** では，その I/O を起動したプロセッサに割り込むため，キャッシュに残っているデータを有効に活用できる可能性がある．**2b-2** は，暇な (アイドルな) プロセッサがあればそのプロセッサに割り込むという方式である．アイドルなプロセッサがない場合は，最も低プライオリティなプロセス (スレッド) を実行しているプロセッサに割り込むようにしている．この方式は，他のプロセスの実行を乱すことが少ないため，最も効率が良さそうにみえる．ただし,「暇な」プロセッサを効率良く見つけるためには特別なハードウェアが必要になる場合もある [10]．今回テストに用いた OSF/1 TOP-1 の場合，HD サブシステムがプログラマブルであるため，ソフトウェア的に実現できている．**2b-3** は，2b-2 の逆で，最も高プライオリティなプロセス (スレッド) を実行しているプロセッサに割り込むという方式である．**2b-4** は，全てのプロセッサに割り込む方式である．各プロセッサで起動される割り込みハンドラのうち，ロックをとれたものが実際の割り込み処理を行なう．最後の 2 つの方式はどうみても効率は悪そうだが，どのくらい性能が落ちるかを比較するために評価の対象としている．

次に，性能比較に用いた方法を説明する．図 3がテストに用いたプログラム iotest の概略である．このプログラムは，指定された間隔をおいてデバイスアクセスを繰り返すだけの単純なものである．テストは，このプログラム iotest を複数個走らせ，終了するまでにかかった実時間を比較することで行なった．測定は，OSF/1 TOP-1 をマルチユーザ・モードで立ち上げ，システムの実行にどうしても必要なデーモン類以外は全て kill した状態で行なった．テストに用いたデバイスは，前章で述べた 3 つのデバイス (scsihd, testdev, ramdisk) である．OS 内のバッファキャッシュの影響をなくすため，全て RAW デバイスでアクセスしてテストしている．

```
/*
 * iotest.c - デバイスアクセスの性能比較のための
 *            テストプログラム
 */
main(ac, av)
int  ac;
char **av;
{
    char *dev_name;
    int  loop_count, wait_count;
    int  fd, i, j, k;
    char buf[4096];

    /* オプションなどのセットアップ */
    dev_name   = <テストするデバイス>;
    loop_count = <ループの回数>;
    wait_count = <ウェイトカウント>;

    fd = open(dev_name, 0);

    /* メインループ */
    for (i = 0; i < loop_count; i++) {

        /* デバイスアクセス */
        read(fd, buf, 4096);

        /* 一定時間待つ */
        for (j = 0; j < wait_count; j++)
            for (k = 0; k < 1000; k++)
                ;
    }

    close(fd);
    exit(0);
}
```

図 3: テストに用いたプログラム

5 評価結果及び検討

本章では，問題 1, 2 それぞれについて，前章で述べたテストの測定結果を示し，検討を行なう．

5.1 デバイスの起動について

表 1は，**問題 1** に関するテスト結果である．ファネリングを行なうかどうかによる性能差は，同時に動いているプロセス数や，デバイスアクセスの頻度によって変動することが予想される．そこで今回のテストでは，同時に実行するプログラム (iotest) の数と，プログラム中のウェイトカウントの値を変化させて比較を行なった[3]．なお，このテストではデバイスからの割り込みは全て PU1(unix_master) に送られる (2a-1) ように設定している．表の各データは，「ファネリングする (1a)」場合に比べて,「ファネリングをやめる (1b)」ことでどれだけ性能が向上したかをパーセント表示で示したものである．なお，scsihd の表で "—" と表示されている部分は，測定ごとのばらつきが大き過ぎて性能差を求めるのが不可能だった部分である．図 4 は，横軸を「同時に動かすプロセス数」，縦軸を性能向上率として表 1のデータをグラフ化したものである．

この表及びグラフからわかる通り，ファネリングをやめてデバイスの起動をどのプロセッサからでも行なえるようにすることで，ある程度の性能向上がはかれるようである．例えば scsihd の場合だと，最大 25%程度の性能向上がみられる．ただし，実行されているプロセスの数がプロセッサの数 (OSF/1 TOP-1 では 9 個) を越えると，ほとんど効果がみられなくなる．これは，プロセスがプロセッサの数より多くなると，複数のプロセスがタイムスライシングによりプロセッサに割り当てられるようになるため，ファネリングによるプロセッサ移動の影響がその

[3] 実測では，ウェイトカウント 10 あたり，ほぼ 20msec の「待ち」が入る．当然，この値が大きいほど，デバイスアクセスの頻度は下がることになる．

(ア) scsihd

ウェイトカウント	プロセス数												
	1	2	3	4	5	6	7	8	9	10	12	15	20
0	0.0	0.5	—	—	—	—	—	—	—	—	—	—	—
10	0.0	25.8	23.1	—	—	—	—	—	—	—	—	—	—
20	0.0	24.4	26.7	26.3	—	—	—	—	—	—	—	—	—
50	0.0	0.7	0.5	0.2	0.5	0.5	—	—	—	—	—	—	—
100	7.2	7.8	7.6	7.6	7.7	7.7	7.5	7.3	4.3	-0.5	0.4	0.2	0.4
200	0.2	0.2	0.6	0.7	0.9	1.1	0.9	0.9	0.5	0.2	0.4	0.0	0.2

(イ) testdev

ウェイトカウント	プロセス数												
	1	2	3	4	5	6	7	8	9	10	12	15	20
0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	-0.7	-0.1
10	1.2	24.4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	-0.3
20	2.2	13.3	15.5	1.4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
50	2.4	3.9	3.5	3.3	4.4	4.8	3.1	0.6	0.6	0.4	0.3	0.1	0.0
100	2.0	2.4	3.0	3.0	3.4	3.0	3.1	3.3	3.0	-0.4	0.7	0.3	0.4
200	1.0	1.0	1.2	1.8	1.7	1.8	1.2	1.0	1.3	0.5	0.1	0.1	0.4

(ウ) ramdisk

ウェイトカウント	プロセス数												
	1	2	3	4	5	6	7	8	9	10	12	15	20
0	5.5	107	200	279	392	460	510	579	611	617	625	627	627
10	8.3	35.0	30.0	34.4	35.5	33.3	35.0	20.7	27.6	28.8	27.2	27.4	27.7
20	7.1	20.7	19.7	21.9	21.9	19.8	19.8	23.0	20.5	6.6	2.0	2.3	2.8
50	5.5	8.4	8.9	9.4	8.7	8.2	7.8	7.8	8.7	0.9	0.0	0.0	0.9
100	3.5	3.3	3.3	3.0	3.2	3.5	3.5	3.4	2.0	-0.4	1.1	0.3	0.6
200	1.9	1.8	1.6	1.9	2.0	2.2	1.9	1.6	1.2	-0.6	0.0	0.0	0.1

表 1: ファネリングするかどうかによる性能差

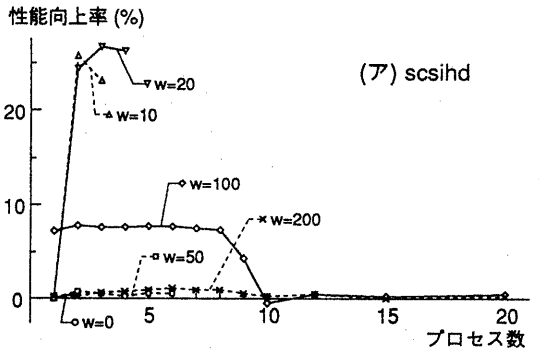
中に隠れてしまうためだと思われる．逆に，プロセスが1個の場合も，効果があまりみられない．この傾向は，アクセスの頻度が高い状況でより顕著である．これは，プロセスが1個で，しかも頻繁にアクセスしている場合，1bの方式では結局そのプロセスは unix_master 上でずっと走ることになるため，ファネリングの際に，実際のプロセッサ移動が必要なくなるためである．プロセスが1個の場合でも，アクセス頻度が下がってくると，プロセスがデーモンプロセスの起動などのタイミングで unix_master から追い出されるため，差が出てくるようである．これらのことから考えて，性能差の主な要因は，ファネリングに伴う無駄なプロセッサ移動にあると考えられる．

アクセス頻度について考えると，当然，頻度が高い方が性能差も大きくなる傾向がみられる．ただし，scsihd の場合，プロセス数が多くなったり，アクセス頻度が高くなると測定ごとのばらつきが大きくなり，性能差を求めることが不可能になってしまう (表1の "—" の部分)．testdev では測定ごとのばらつきはそれほどみられないが，同様の領域でほとんど性能差があらわれなくなっている．この領域でのシステムの挙動をモニタしてみると，デバイスの処理が飽和してしまっていることがわかる．つまり，デバイス処理が飽和してしまう領域では，「デバイス・ボトルネック」が生じてしまい，ファネリングによる性能差はあまり観測できなくなるということである．scsihd でばらつきが大きくなってしまうのは，ディスクのヘッドスケジューリングが微妙なタイミングでうまくいったりいかなかったりするためだと思われる．scsihd 用デバイスドライバの場合，このヘッドスケジューリングの関係で，ロックによる排他制御が必要な時間が比較的長くなっている．そのため，1a の方式でアクセス頻度が高い状況ではロックによるオーバーヘッドが増加している可能性があるが，今回のテストでは測定ごとのばらつきの方が大きくなってしまい，うまく観測できなかった．また，scsihd ではウェイトカウントが 50 の場合だけ，ほとんど差がみられなくなっている．これもヘッドスケジューリングのタイミングが原因だと思われるが，詳細は今後の調査が必要である．

ramdisk では，他の2つのデバイスとは少し異なった現象がみられる．つまり，アクセス頻度が高い領域では逆に性能差が激しくあらわれている．この状況をモニタしてみると，ファネリングしている場合，unix_master での処理が飽和してしまっていることがわかる．ファネリングしない場合は，デバイスアクセスの処理が各プロセッサに分散されるので，性能が悪化しない．つまり，ramdisk のように，プロセッサでの処理が多いデバイスでは，ファネリングすると「デバイスアクセス・ボトルネック」という状況が生じてしまい性能が著しく悪くなるということである．

割り込みの方式	プロセス数												
	1	2	3	4	5	6	7	8	9	10	12	15	20
2a-1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2a-2	0.0	0.0	0.0	0.0	-0.3	-0.5	-1.4	-2.0	-0.5	0.3	0.1	0.0	0.1
2b-1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	-0.1	0.4	0.5	0.1	0.2	0.1
2b-2	0.0	-0.2	0.0	0.1	-0.1	0.0	0.0	-0.2	0.4	0.1	-0.1	-0.2	0.2
2b-3	0.0	0.0	0.0	0.1	-0.4	-1.1	-1.9	-2.0	-0.2	-0.1	-0.1	0.2	0.2
2b-4	0.0	-0.1	0.0	0.0	-0.1	-0.3	-1.3	-3.3	-2.8	-1.0	-1.2	-0.9	-1.0

表 2: 割り込みの配送方式による性能差



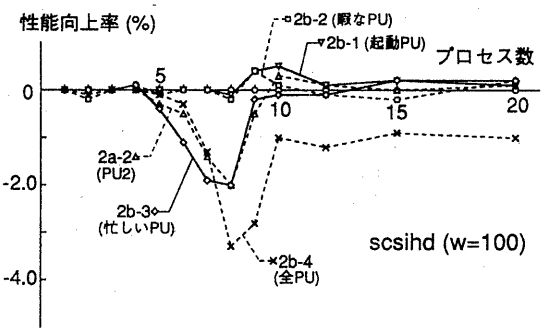
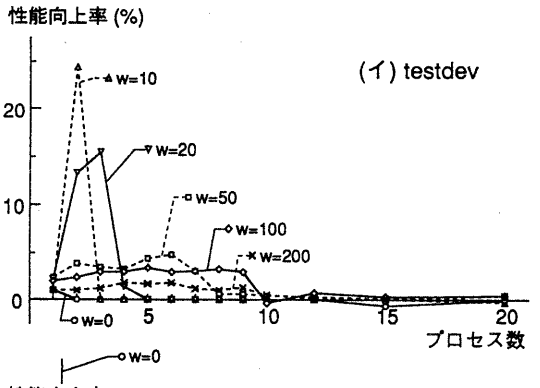


図 5: 割り込みの配送方式による性能差



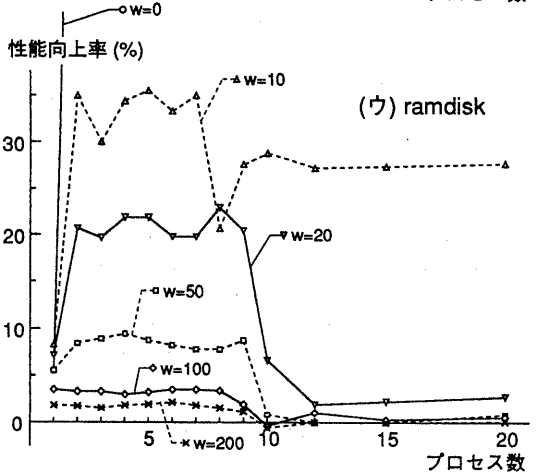


図 4: ファネリングするかどうかによる性能差

5.2 割り込みの配送について

次に，**問題 2** の割り込みの配送について検討する．表 2は，問題 2 に関するテスト結果を示したものである．表の各データは，「常に PU1(unix_master) に割り込んだ場合 (2a-1)」に比べて，それぞれの方式でどれだけ性能が向上したかをパーセント表示で示したものである．なお，このテストではデバイスの起動には 1a の対称な方式を用いている．テストは scsihd で行ない，ウェイトカウントには問題 1 で比較的安定した結果の得られた 100 を指定している．図 5は，横軸を「同時に動かすプロセス数」，縦軸を性能向上率として表 2のデータをグラフ化したものである．この表及びグラフからわかる通り，

2b-4 < 2b-3 < 2a-2 < 2a-1 ≅ 2b-1 ≅ 2b-2

の順で性能が良くなるようである．

2b-4 の全プロセッサに割り込む方式は，直観的にも性能が悪いのは明らかであろう．しかし，その場合でも最大 3%程度の差しかみられない．問題 1 でみられた性能差に比べればほとんど無視できる範囲ともいえる．2b-4 以外では，プロセス数がプロセッサ数より多くなると差がみられなくなる．これは，性能差の主原因が「暇なプロセッサに割り込んでいるかどうか」にあるためだと考えられる．「忙しい」プロセッサに割り込んでしまうと，そこで動いていたプロセスが割り込みによって中断されてしまうため性能が悪くなる．プロセス数がプロセッサ数より多くなると，「暇な」プロセッサが存在しなくなるため，どの方式でもほぼ同じ結果になるようである．

2b-3 の方式は，常に「忙しい」プロセッサに割り込むため性能が悪くなっている．特に，プロセス数が5～7では，2b-4より悪い結果が出ている．これはおそらく，2b-4では全プロセッサがロックをとりあった結果,「暇な」プロセッサで割り込み処理が行なわれることがあるためと思われる．

2a-2 の，PU2に割り込む方式では，割り込んだ時たまたまPU2でプロセスが走っていた場合にペナルティが課せられる．プロセス数が5～7あたりで2b-3より良い結果になっているのは,「たまたまPU2でプロセスが走っている」確率のためだと考えられる．

一方，同じ特定プロセッサに割り込む方式でも，**2a-1** のPU1(unix_master) に割り込む方式では性能の悪化がみられない．これは，OSF/1のスケジューリング方式のためである．OSF/1では，プロセス(スレッド)数がプロセッサ数より少ない場合，なるべくunix_masterを空けておくようなスケジューリングを行なっている．つまり，2a-1の方式は結果的に「暇な」プロセッサに割り込んでいることになるため，性能が悪化しないということである．

2b-1 の，起動プロセッサに割り込む方式も，性能の悪化がみられない．これは，プロセス数がプロセッサ数より少ない場合，I/Oを起動したプロセッサはそのI/Oが終了するまで「暇に」なっているためである．

2b-2 では積極的に「暇な」プロセッサに割り込むようになっているため，性能が悪化しないのは当然ともいえる．今回のテストは，同じプログラムを複数走らせることで行なったため，結果的に2a-1や2b-1との差がほとんどみられないが，多種多様なプロセスが動いている実際の処理環境では，もっと性能が向上する可能性もある．

6 おわりに

本稿では，MP UNIXシステムのデバイスアクセスについて，プロセッサを固定した時と並列化した時の性能差を中心に調べた．デバイスドライバを並列化し，どのプロセッサでもデバイスアクセスを行なえるようにすることは，全体的にある程度効果があるようである．ただし，効果の程度はデバイスのタイプや，システムの負荷状況によってかなり変動がみられた．一方，デバイスからの割り込みの配送を対称化することは，少なくともOSF/1の場合にはそれほど大きな効果がみられなかった．

ただし，今回テストに用いた方法はかなり単純化されたもので，実際の処理環境とは異なる部分がある．今後，もっと現実の環境に近い状態でのテストも行なっていく予定である．また，今回はプログラムの実行時間という巨視的なデータで比較を行なったが，キャッシュのヒット率や，ロックの効率といった微視的な部分の調査も必要であろう．なお本稿ではふれなかったが，システム全体の性能という点でみた場合，デバイスドライバ以外の部分，例えばスケジューラや，デバイスそのものの性能も非常に重要である．今後はこれらの点についても考えていきたい．

謝辞

本研究の機会を与えて下さった日本IBM東京基礎研究所の鈴木則久所長と天明崇氏に感謝します．また，OSF/1のTOP-1への移植にあたり，開発環境であるODE (OSF Development Environment) を整備して下さった緒方正暢氏と，日頃議論していただいている森山孝男氏に感謝します．

参考文献

[1] 河内谷清久仁 他: "TOP-1 オペレーティング・システムの構造," 情報処理学会研究報告 90-OS-48, pp.25–34 (1990)

[2] N. Yamanouchi: "Performance Effects of Program Structures on a Snoop-Cached Multiprocessor System," Proc. of InfoJapan '90, IPSJ, pp.339–346 (1990)

[3] D. Patterson, et al.: "A Case for Redundant Arrays of Inexpensive Disks (RAID)," Proc. of ACM SIGMOD Conf., Chicago, IL., pp.109–116 (1988)

[4] N. Oba, et al.: "TOP-1: A Snoop-Cache-Based Multiprocessor," Proc. of the 9th. Annual IEEE International Phoenix Conf. on Computers and Communications, pp.101–108 (1990)

[5] S. Shimizu, et al.: "Design and Evaluation of Snoop-Cache-Based Multiprocessor, TOP-1," Proc. of the International Symposium on Shared Memory Multiprocessing, pp.209–217 (1991)

[6] OSF: "Symmetrical Multiprocessing in the OSF/1 Operating System," OSF/1 White Paper, OSF-OSMP-WP13-1190-1 (1990)

[7] M. Accetta, et al.: "Mach: A New Kernel Foundation for UNIX Development," Proc. of USENIX 1986 Summer Conf., pp.93–112 (1986)

[8] OSF: "The Design of the OSF/1 Operating System, Chapter 18. I/O Subsystem," OSF/1 Document, OSF-ED-OSD-1290-1 (1990)

[9] OSF: "OSF/1 System Extension Guide, Chapter 8. Multiprocessing Under OSF/1," OSF/1 Document, OSF-ED-OSXG-1290-1 (1990)

[10] B. Beck, et al.: "VLSI Assist For A Multiprocessor," Proc. of the 2nd. International Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS-II), pp.10–20 (1987)