情報処理学会論文誌　コンピューティングシステム　Vol. 4　No. 2　1–11 (Mar. 2011)

仮想計算機における仮想ディスプレイの処理負荷低減法

後 藤 真 孝[†1]　峰 松 美 佳[†1]

近年，計算機の仮想化を利用した仮想デスクトップ環境が普及しつつある．オペレーティングシステムやアプリケーションを改変することなく計算機資源の集約を実現できる一方で，実計算機の処理に比べ処理速度の低下が問題である．特に，仮想フレームバッファに描画されたグラフィックの表示性能劣化が顕著であり，ユーザの利便性低下に与える影響が大きい．そこで我々は，メモリ管理ユニット（MMU）のメモリ更新検出機能の使用を前提に，仮想フレームバッファの転送処理負荷の低減を行った．MMU のページフレームのサイズと対応したタイルでフレームバッファを構成する，ページタイルフレームバッファを考案し，画面の更新検出処理時間の低減を図った．また，MMU のページテーブルの更新ビットを使用したページテーブル検索方式と，メモリ書き込み例外を使用した書き込み例外方式について，試作を行って性能を比較した．性能比較実験により，評価で用いた描画パターンにおいて，ページタイルフレームバッファを用いた場合に画面の更新検出処理時間を 10%程度削減していることを確認した．また，ページテーブル検索方式と書き込み例外方式については，仮想計算機における仮想ディスプレイの更新検出性能の視点ではフレームレート，CPU 使用率に有意な差はなかった．

Optimization Method for Virtual Display Device of VM System

Masataka Goto[†1] and Mika Minematsu[†1]

Recently, the use of virtual desktop infrastructure which provides virtual machines has spread. It enables efficient use of computer resources without OS or application modification, while processing performance is less efficient than a physical machine. The display performance of graphic drawn in the virtual frame buffer especially deteriorates, which has a great influence on usability. Thus we have improved the processing performance of virtual frame buffer assuming the use of memory update detection function of MMU (Memory Management Unit). We propose page tiled frame buffer which constructs frame buffer with tiles that corresponds to page frame size of MMU to reduce the screen update detection time. Furthermore, we propose page table lookup method and write exception method, which utilizes dirty bits of page tables and write exception of MMU for update detection, respectively. We exhibit the performance comparison of the two methods using prototype implementations. The results of performance comparison evaluation show that page tiled frame buffer reduces screen update detection time by 10% for the draw patterns used in the evaluation. Furthermore, frame rate and cpu utilization of page table lookup method and write exception method show no significant difference in terms of update detection performance by virtual display device.

1. はじめに

近年，計算機性能の高度化により，計算機資源の集約が可能となってきている．ネットワークの低遅延化や広帯域化の恩恵を受け，エンドユーザのデスクトップ環境の集約が図られるようになった．このデスクトップ環境の集約にはには様々な手法があるが，計算機の仮想化を利用した仮想デスクトップ環境（VDI）は，既存のオペレーティングシステム（OS）やアプリケーションの改変が不要であるため，有力な方法の 1 つである．このため，多くの仮想計算機実装は，VDI の提供を念頭においた機能を有している．

レガシーシステムの維持を多くの局面で必要としつつ，デスクトップ集約の恩恵を必要としている一例として社会インフラの制御システムがある．社会インフラシステムの制御センタでは，限られた人数の管理者で広範囲のインフラを運用管理するために，マルチモニタ構成の計算機で様々な作業を実施している．個々の計算機が独立したシステムとして動作している場合もあるため，管理者の机上のモニタやキーボードに多数の計算機を集約するために，キーボード/ビデオ/マウス切替え器（KVM スイッチ）を用いている場合もある．しかしながら，計算機台数の増加から管理者の机上との距離の延伸が起き，自在な表示関係を実現するための KVM ケーブルの複雑化の問題が発生している．このような多数の計算機の集約と自在な KVM の延長を，既存の OS やアプリケーションの改変なく実現できる点において，仮想計算機による VDI は社会インフラの制御システムに適している．

図 1 に，仮想計算機サーバによる VDI のシステム構成例を示す．仮想計算機サーバのゲストドメインに既存のシステム運用アプリケーションを動作させ，ユーザ端末からネットワークを介した仮想デスクトップとしてアプリケーションを利用するシステムである．仮想計算機は実計算機の処理に比べ処理速度の低下が起こりがちであり，特に仮想フレームバッ

†1 株式会社東芝研究開発センター
R&D Center, TOSHIBA Corporation

ファを使った VDI は性能面に難点があるが，構造が単純で，いろいろなゲスト OS に対応するためのコストを低く抑えることができるという利点がある．そこで本稿では仮想フレームバッファの表示性能を改善し，より実用的な速度で動作させるための方式を提案する．

我々はこれまでに，ネットワークを用いた画面転送についてプロトコルや端末の研究開発を行ってきた[1),2)]．仮想計算機の仮想フレームバッファの表示性能に関しては，仮想フレームバッファの画面の更新部分の検出処理に着目し，メモリ管理ユニット（MMU）のメモリ更新検出機能の使用を前提に，MMU のページフレームのサイズと対応したタイルでフレームバッファを構成する手法（ページタイルフレームバッファ）を提案している[3),4)]．

本稿では，ページタイルフレームバッファの方式を述べ，MMU の 2 通りの使用方法であるページテーブルの更新ビットを使用したページテーブル検索方式と，メモリ書き込み例外を使用した書き込み例外方式について述べる．また，簡易試作を評価し，今回用いた描画パターンにおいては，画面の更新検出処理時間を 10%程度削減していることを述べる．さらに，ページテーブル検索方式と書き込み例外方式については，本稿の評価視点においては有意な差はなかったことを示す．

2. 仮想計算機の画面転送処理の構成と課題

2.1 仮想計算機の画面転送処理の構成と問題点

図 2 に仮想計算機の画面転送処理の構成を示す．仮想計算機サーバは，実計算機を仮想的に複数の計算機（ドメイン）として利用できるようにする VMM と，VMM のコントロールを行う管理ドメイン，ユーザの仮想計算機が動作するゲストドメインからなる．ゲストドメインにはゲスト OS が動作し，ユーザアプリケーションはゲスト OS 上のアプリケーションプログラムとして動作する．仮想計算機の仮想ディスプレイは，VMM の仮想フレームバッファと管理ドメインのデバイスエミュレータで実現される．

仮想フレームバッファは，ゲストドメインと管理ドメインの間で共有される共有メモリとして作成される．アプリケーションプログラムが直接，あるいはグラフィックライブラリを介して間接に仮想フレームバッファにピクセルデータを書き込む．近年は，グラフィック効果を有効に利用したアプリケーションが増え，ビデオコントローラの描画機能を使用したり，グラフィック処理ユニット（GPU）を用いて描画を行ったりするものが増えてきている．一般的な仮想計算機環境では，管理ドメインで動作しているデバイスエミュレータの仮

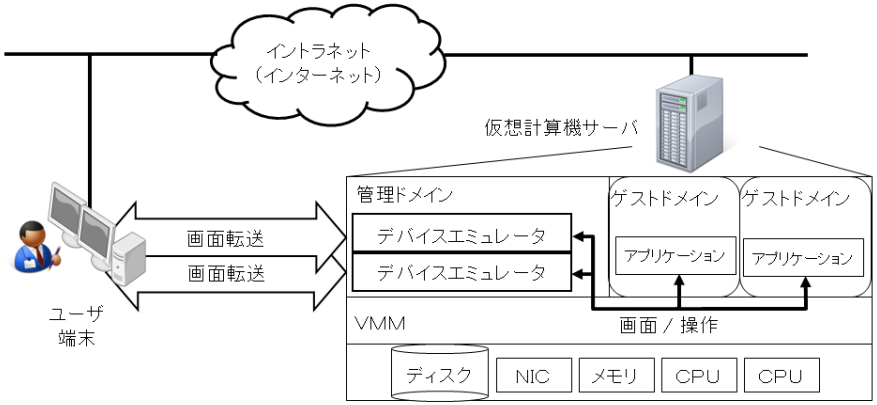


図 1　仮想計算機サーバのシステム構成例
Fig. 1　Example of VM server system.

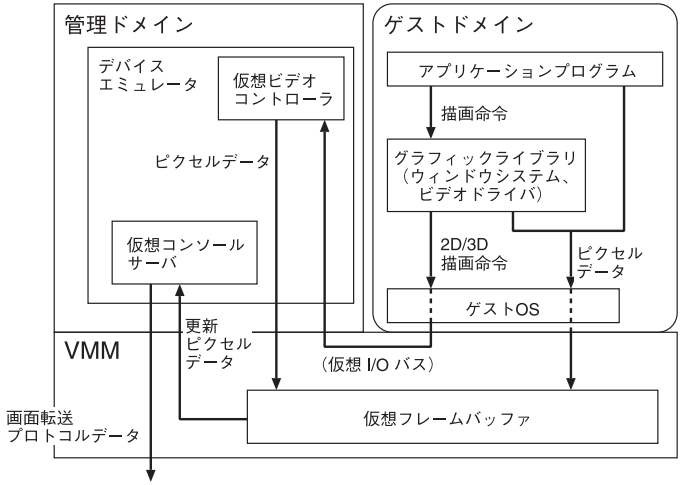


図 2　仮想計算機の画面転送処理の構成
Fig. 2　Screen image transfer of VM.

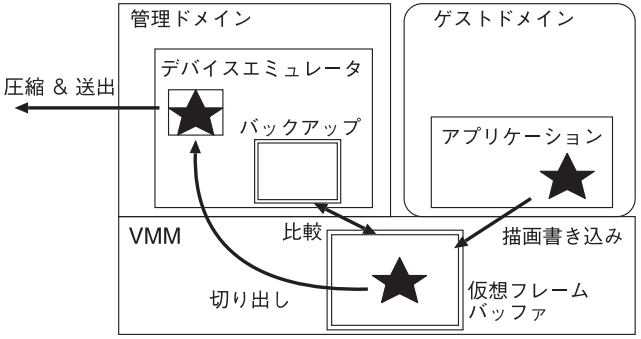


図 3　画面転送の例
Fig. 3　Example of screen image transfer.

想ビデオコントローラが，それらの描画を仮想的に代行し，仮想フレームバッファへピクセルデータとして書き込みを行う．

その後，デバイスエミュレータの仮想コンソールサーバは，仮想フレームバッファの内容の変化を更新ピクセルデータとして検出する．仮想コンソールサーバは，検出した変化画像を圧縮し，画面転送プロトコルデータとして利用者端末に伝送する．以上により，仮想計算機のコンソールの画面転送が実現される．遠隔操作プロトコルとしては，VNC[6] や RDP[5] が有名である．

図 3 には，上述の処理の流れを簡易化し，具体的な描画例を用いて示している．ゲストドメインで動作するアプリケーションが画面上に星のイメージを描画する場合の例である．まず，アプリケーションが仮想フレームバッファに，描画内容である星のピクセルデータの書き込みを行う．管理ドメインで動作するデバイスエミュレータは，あらかじめ決められた周期，たとえば 30 ms で，画面の更新検出処理を行って，更新ピクセルデータの取得を行う．更新検出処理とは，直前のフレームバッファのバックアップと現時点の仮想フレームバッファをピクセルごとにバイト比較し，データの変化の有無を検出し，変化があった矩形の切り出しを行う処理である．切り出された矩形は画像圧縮され，ユーザ端末へ送出される．

以上のように，仮想フレームバッファによる VDI は，デバイスエミュレータが周期的に仮想フレームバッファ全体に対して更新の有無を確認して実現されている．このため，仮想計算機の台数に比例して，管理ドメインに余計なオーバヘッドがかかるという問題があった．しかし，周期はユーザ側の画面のフレームレートに影響するため不用意に広げるわけにはいかず，また，更新の有無の確認の精度は圧縮対象の画像サイズに影響し，画像サイズの増大は圧縮処理の高負荷につながるため，不用意に省略するわけにはいかない．このように，効果的な画面の更新検出処理の必要性があった．

2.2　関連研究と本研究の課題

画面転送の高速化手法に関する研究はいくつかある．まず，画面の描画に合わせた画像データの圧縮に着目し，伝送するデータ量を削減してネットワークの影響を小さくする手法がある[7]．また，スクロールやウィンドウの移動など画面描画でよく用いられる領域コピーや領域移動の描画を描画命令の種別をもとに検出し，領域コピーや領域移動の描画命令自身を伝送して，表示端末側で描画命令に従った表示を行う手法もある[8]．これらの方式は，描画内容の把握や特定の描画命令を検出する必要があるため，図 2 のグラフィックライブラリのように，アプリケーションプログラムから抽象化された描画命令を取得するブロックで画面を転送する場合に適している．しかし，様々なゲスト OS などの混在があることを考えると，1 つのプロトコルで KVM を統合できる仮想計算機の VDI の利点を損なう恐れがある．

また，仮想計算機の 3D グラフィックの高速化として，OpenGL[9] 対応のアプリケーションが行う 3D グラフィックの描画命令を表示端末に直接伝送する手法が提案されている[10]．この方式は，グラフィック描画をユーザ端末側で行うため仮想計算機サーバ側の処理負荷軽減効果と，画面の転送レイテンシを抑える効果が期待できる．この手法は，2D グラフィック描画への応用についても可能であると考えられる．しかし，アプリケーションが混在する場合もあり，すべての描画に対応するには多くの工数が必要となりうる．また，ユーザ端末への画面転送が，アプリケーション実行中に開始される場合，全画面のリフレッシュ画像データを保持しておく必要があるため，現実的には仮想計算機サーバによる仮想フレームバッファへの画面描画は省略できない．

一方，特定の描画によらない VMM による画面転送の高速化手法もいくつかある．仮想計算機の I/O 全般の高速化を対象とする準仮想化[11] は，仮想計算機用の仮想デバイスを再定義して，仮想ディスプレイの制御手順を定めて性能を高める方式である．仮想デバイスを再定義するためゲスト OS への制約が生まれ，汎用性や簡単さという面では完全仮想化の仮想フレームバッファには及ばない．

完全仮想化に関する手法としては，Huang らが，メモリ管理ユニット（MMU）を用いたオンデマンドページングの方式と同等の手法を採用して処理負荷の軽減を図る方式を提案している．仮想ディスプレイの画面の更新検出を，ゲストドメインと管理ドメインの間で共有されたメモリへのデータ書き込みの検出であるととらえ，仮想メモリ空間機能を持つ多く

の OS で行われているように，仮想フレームバッファの更新ピクセルデータの取得に MMU のページテーブルの更新ビットを用い，4 KiB などのサイズを単位としたページフレームごとにデータ書き込みの有無を検出する方式である．以降，この方式をページテーブル検索方式と呼ぶ．Huang らは，このページテーブル検索方式を Xen の仮想ディスプレイに適用して評価を行っている[12]．従来のピクセルごとのバイト比較では，ゲストドメイン数に比例して管理ドメインの CPU 利用率が高まっていたが，MMU を使用することでゲストドメイン数には影響を受けず，一定の CPU 利用率となることが示されている．

しかし，処理負荷が単純にゲストドメイン数には比例しなくても，アプリケーションからの描画量が増えれば，更新ビットにより書き込まれたと判定されるページフレーム数が増え，仮想ディスプレイに関する処理負荷は増大する．従来の仮想ディスプレイは，フレームバッファ中の個々のピクセルデータがラスタ方向に線形にアドレッシングされているため，ページフレームが 4 KiB で 32 bpp である場合，ラスタ方向への 1024 ピクセルの直線がページフレームに相当する．このため，**図 4** に示すように，ある図形を画面に描画した場合，実際に描画された図形のデータ量に比べ，データ更新として検出されるページフレーム数が多くなりすぎるという問題が依然残されている．

一方，MMU は書き込み禁止のページフレームへのデータの書き込み時に例外を発生させる方法で，特定のページフレームへのデータの書き込みの瞬間をとらえる機能も提供する．フレームバッファを構成するページフレームを書き込み禁止状態にし，アプリケーションが描画データを書き込んだ際にデータ書き込み例外により書き込みを検出できる．以降，この方式を書き込み例外方式と呼ぶ．一般的には，メモリへのデータの書き込みを検出する場合，多くの回数の例外を発生させる書き込み例外方式よりも，ソフトウェアによる処理オーバヘッドがないページテーブル検索方式を用いる方が，処理負荷軽減の効果が大きい．しかし，仮想計算機ではシャドーページテーブルなどの手法により MMU 自身の機能も仮想化されているため，一概にページテーブル検索方式の方が処理負荷軽減効果が高いとは限らない．

以上より，我々は，画面の更新検出時の更新検出ページフレーム数の削減と，ページテーブル検索方式と書き込み例外方式の性能差を明らかにすることを課題とした．次章では，更新検出ページフレーム数削減の提案方式であるページタイルフレームバッファについて述べ，ページタイルフレームバッファを用いた場合のページテーブル検索方式と書き込み例外方式の両方式の処理の流れについて述べる．

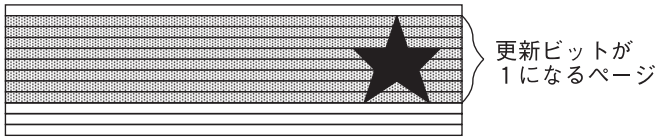


図 4　リニアフレームバッファの更新検出
Fig. 4　Update detection of linear frame buffer.

3. 更新検出ページフレーム数削減法

3.1 ページタイルフレームバッファ

図 4 に示した従来のフレームバッファ構造（以後，リニアフレームバッファと呼ぶ）では，たとえば，64 ピクセル四方のアイコンを画面に描画した場合，更新を検出するページフレーム数は 64 ページとなる．更新検出される 64 ページフレーム（256 KiB）に対し，実際に書き込まれたデータ量（16 KiB）の比率を考えると，有益な検出割合は 6.25%にとどまる．

そこで，我々は，**図 5** に示すように，画面の矩形をタイルと見立て，そのタイルを敷き詰めて画面を構成するページタイルフレームバッファを考案した．ページタイルフレームバッファは，タイル内でアドレスが線形となるようにし，1 タイルのバイトサイズをページフレームと一致させる．たとえば，32 bpp で 32 ピクセル四方を 1 タイルとする．この場合，先の例と同様に 64 ピクセル四方のアイコンを描画した場合，更新を検出するページフレーム数はたかだか 9 ページフレームとなる．更新検出される 9 ページフレーム（36 KiB）に対し，実際に書き込まれたデータ量（16 KiB）の比率を考えると，有益な検出割合は，最悪でも 55.55%になり，この正方形のアイコン描画の例では，有益な検出割合は 7 倍以上改善される．

ただし，ウィンドウ下部の枠の描画のような，高さのないきわめて横長の描画に関しては，リニアフレームバッファの方が有利な場合もある．しかし，通常の大部分の画面描画

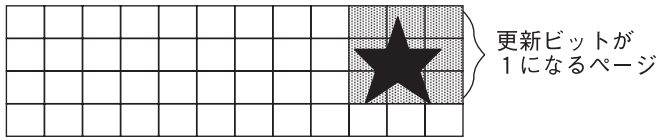


図 5　ページタイルフレームバッファの更新検出
Fig. 5　Update detection of page tiled frame buffer.

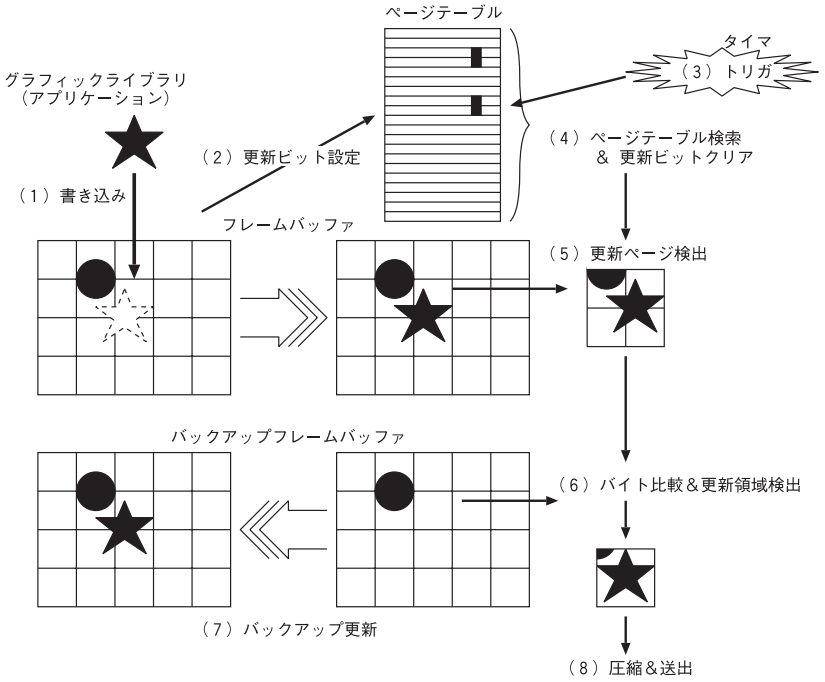


図 6　ページテーブル検索方式
Fig. 6　Page table lookup method.

は，ウィンドウやアイコン，ポインタ，フォントなどのように矩形の形状が多く，ページタイルフレームバッファが効果的であると期待できる．

3.2　ページテーブル検索方式

図 6 に，ページタイルフレームバッファを用いた場合のページテーブル検索方式の流れを示す．ページテーブル検索方式は，ページテーブルの更新ビットを参照することにより，画面中の更新領域を検出する方式である．まず，あらかじめフレームバッファ全体のバックアップを保持しておく．次に，アプリケーションが星の画面表示のために，フレームバッファへの描画を行う（図中 (1)）．MMU は，描画によるメモリへの書き込みに従い，ページテーブル中の当該ページフレームのエントリの更新ビットを設定する（図中 (2)）．仮想計算機における MMU の動作は，VMM が仮想計算機対応にエミュレートして実現する．

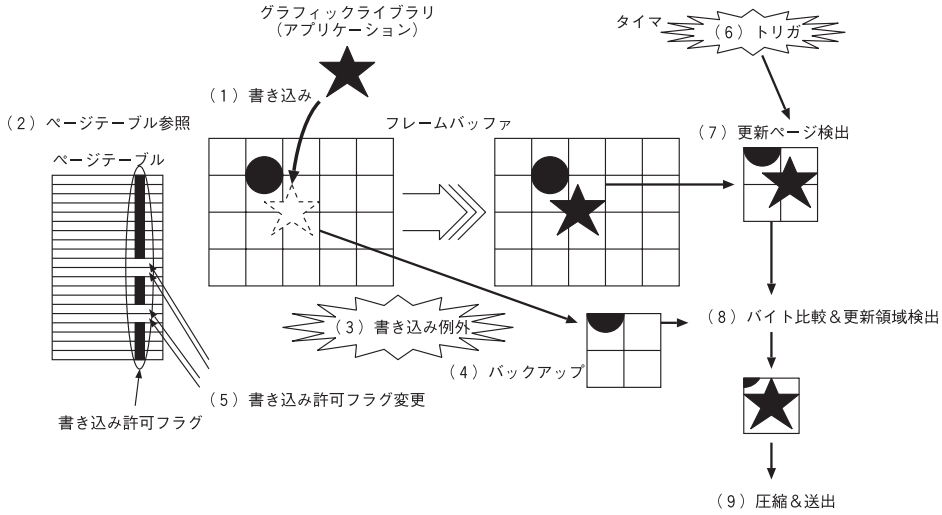


図 7　書き込み例外方式
Fig. 7　Write exception method.

次に，仮想ディスプレイの画面転送処理は，タイマによるトリガを契機に（図中 (3)）ページテーブルの更新ビットの検索を行い，書き込みが行われたフレームバッファ中のページフレームを同定する，と同時に更新ビットのクリアも行う（図中 (4)，(5)）．ここで，書き込みが検出されたページフレームの内容について，現在の内容とバックアップの内容の間でバイト比較を行う（図中 (6)）．これは，たとえば，元々黒であったピクセルに黒を上書きした場合など，同色の上書き領域を画像圧縮の領域からあらかじめ排除して圧縮の処理負荷を軽減するためである．MMU の更新ビットは，書き込みの有無の真偽を判定できるのみである．無駄な画像圧縮と送出を省略するには，実際にデータが改変されたかどうかの判定が必要であり，書き込みが検出されたページフレームに対して，新旧のバイト比較を要する．

そして，フレームバッファのバックアップを更新し（図中 (7)），実際にデータが改変された部分に関して，ユーザ端末への画面データの圧縮と送出を行う（図中 (8)）．

3.3　書き込み例外方式

図 7 に，ページタイルフレームバッファを用いた場合の書き込み例外方式の流れを示す．書き込み例外方式は，書き込み時の例外を利用して画面中の更新領域を検出する方式である．まず，あらかじめフレームバッファ全体のページフレームに関して，ページテーブルの

エントリ中の書き込み許可フラグを不可にしておく．

次にアプリケーションが星の画面表示のために，フレームバッファへの描画を行う（図中 (1)）．MMU は，書き込み禁止のページフレームに対するメモリ書き込みを検出して例外を発生させる（図中 (2)，(3)）．仮想計算機においては，VMM 中の例外ハンドラに処理が移る．例外ハンドラでは，仮想ディスプレイの画面転送処理のために，当該ページフレームのバックアップを保存し（図中 (4)），当該ページフレームの書き込み許可フラグを許可に変更する（図中 (5)）．ここで書き込み許可フラグを許可をすることで，連続する当該ページフレームへの書き込みのつど，書き込み例外が発生することを回避する．図では，4 ページフレームを 1 度にバックアップしているように見受けられるが，実際は各々のページフレームへ 1 回ずつ書き込み例外が発生し，そのつどバックアップを行う．

その後，仮想ディスプレイの画面転送処理は，タイマによるトリガを契機に（図中 (6)），書き込みが行われたフレームバッファ中のページフレームに対して現在とバックアップのバイト比較を行い（図中 (7)，(8)），実際にデータが改変された部分に関してユーザ端末への画面データの送出を行う（図中 (9)）．

4. 実装および評価方法

4.1 実 装 方 針

計算機仮想化には Xen を用いることとした．また，ゲスト OS は Linux を用い，アプリケーションは X window ベースのものを想定したため，ウィンドウシステムとして Xorg を用いることとした．仮想フレームバッファのページタイル化対応は，当初，デバイスエミュレータ内の仮想ビデオコントローラの描画ですべて対応することを考えていたが，画像イメージの put のように，デバイスエミュレータのページタイル化では対応しきれない描画が存在したため，今回の実装では描画のページタイル化はすべて Xorg で施すものとした．

図 8 に，今回の実装イメージを示す．実行時には，ページテーブル検索方式か書き込み例外方式の一方が機能する．また，**図 9** には，ページテーブル検索方式と書き込み例外方式を対比しながら，処理の流れを示した．点線は，2 方式の同一の意味合いの処理の関係を示している．

ページタイル化された Xorg によって画面描画が行われると，ページタイルフレームバッファへの書き込みアクセスが発生する．ページテーブル検索方式では，VMM の処理により，該当するページフレームに対応するページテーブル中のエントリの更新ビットが設定される．書き込み例外方式では，該当するページフレームへの書き込みが許可されていない，

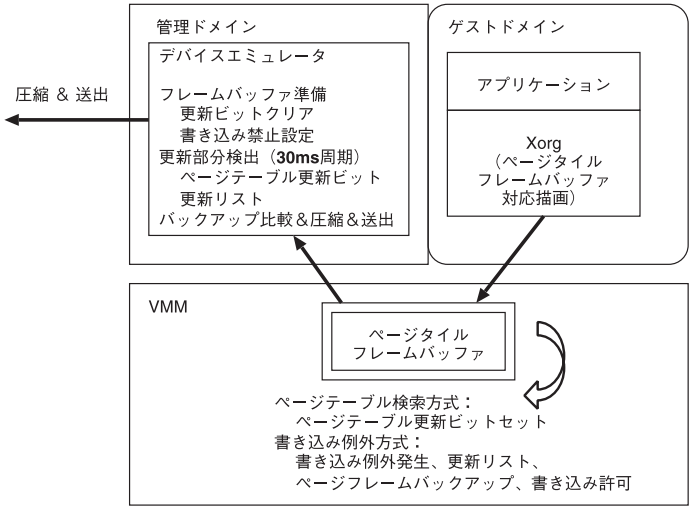


図 8　実装イメージ
Fig. 8　Implementation image.

つまり，初回である場合には書き込み例外が発生する．書き込み例外処理では，書き込みが行われたページフレームをリストで記憶し，当該ページフレームへの書き込みを許可する．

一方，デバイスエミュレータは，30 ms の周期で画面更新検出処理と，更新部分の送出処理を実施する．30 ms 周期のタイマを契機に，更新されたページフレームの把握と圧縮，送信を行って，更新されたページフレームの後かたづけを実施する．ページテーブル検索方式では，ページテーブルの更新ビットの検索で更新されたページフレームの把握を行い，ページテーブルの更新ビットのクリアで後かたづけを行う．一方，書き込み例外方式では，書き込み例外処理で記録された更新ページフレームのリストを参照して更新されたページフレームの把握を行い，更新ページフレームのリストのクリアとページテーブルの書き込み許可クリアで後かたづけを行う．

4.2 試　　作

仮想計算機環境には，Xen3.1 の完全仮想化モードを使用した．ゲストドメインで動作するゲスト OS には FedoraCore6 を使用し，画面の描画には Xorg(X11R7.1) と X アプリケーションを使用した．Xorg のページタイル化対応は，シャドーフレームバッファライブラリのフレームバッファへのデータコピーコードに C 言語で約 110 行のコード追加で対応

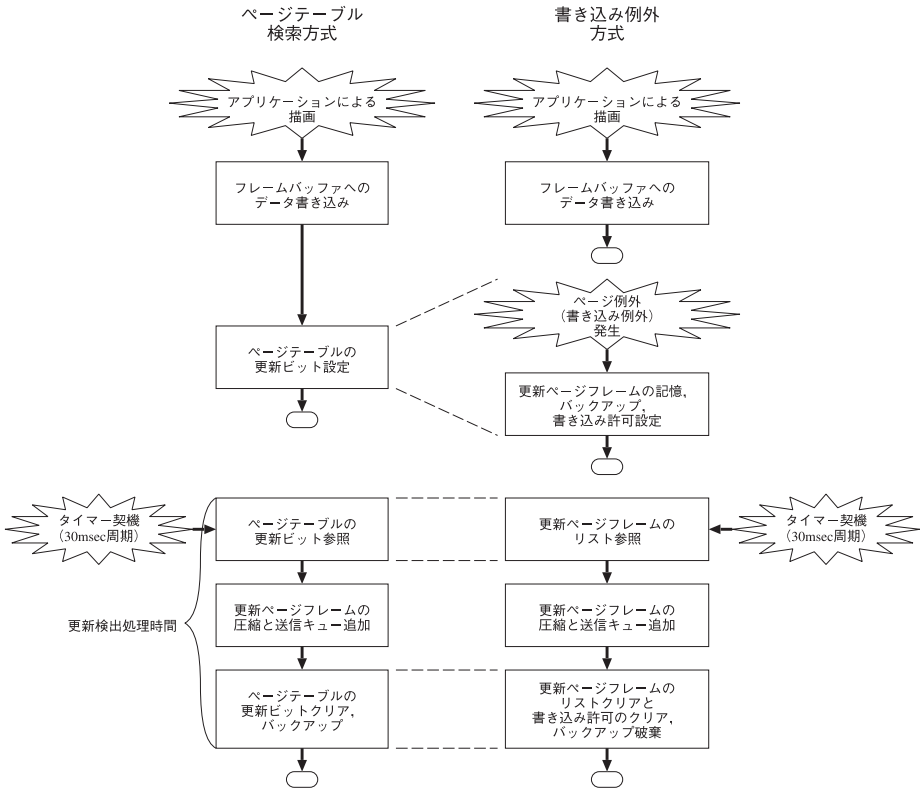


図 9　ページテーブル検索方式と書き込み例外方式の処理の流れ
Fig. 9　Procedure flow.

した．また，管理ドメインのデバイスエミュレータに C 言語で約 600 行程の改変と，ゲストドメインで使用する VGA BIOS が保持しているフレームバッファサイズの定数を変更した．

ページテーブル検索方式の実装は，Huang らによって Xen 開発者メイリングリストに報告された vga-acc.patch [13] を使用した[⋆1]．

⋆1 一部，不具合修正を施した．

表 1　仮想計算機サーバのスペック
Table 1　VM server specifications.

CPU	Dual-Core Intel Xeon 1.6 GHz x2
メインメモリ	4 GB
仮想計算機実装	Xen3.1 + Fedora 7
仮想計算機数	1 台（1 VCPU）
ゲスト OS	Fedora Core 6
VM メモリ	250 MB

書き込み例外方式は，ページテーブル検索方式のインタフェースを踏襲し，例外処理中にゲストドメイン対応に更新ページフレームをビットマップとして記憶する改変を，VMM の内部に対して行った．管理ドメインのデバイスエミュレータが更新検出に関する初期化，および，更新ページフレームビットマップの取得を行うには，ページテーブル検索方式と同じく，VMM インタフェースとして `xc_shadow_vram_control` 関数を使用した．コードの改変量は，それぞれ，管理ドメインのデバイスエミュレータに約 150 行，VMM の例外ハンドラに約 300 行，VMM のゲストドメイン生成処理部などに約 400 行程度行っている．

4.3　評 価 方 法

図 1 の構成で動作させ，仮想計算機サーバ内部の処理時間やフレームレート，CPU 使用率，Xen 統計情報を測定した．

表 1 に使用した仮想計算機サーバを示した．ネットワークは，100BaseTX の Ethernet を用い，仮想計算機サーバとユーザ端末は同一セグメントとした．評価は仮想計算機サーバ上で仮想計算機 1 台を動作させ，ユーザ端末で仮想計算機の仮想ディスプレイを表示して行った．仮想ディスプレイの画面サイズは 1024 ピクセル × 768 ピクセルである．

評価は，ページタイルフレームバッファの効果の評価と，ページテーブル検索方式と書き込み例外方式の比較の評価を個別に行った．また，汎用性という視点で改変箇所に関する利害の考察を行った．

5. 評　　価

5.1　ページタイルフレームバッファの効果

まず，ページタイルフレームバッファを用いて更新検出対象であるページフレームを削減した結果，更新検出処理時間に与える効果を確認した．

評価は 1 台の仮想計算機で hdbench clone を動作させ，リニアフレームバッファとページタイルフレームバッファの双方について，画面の更新検出処理時間（msec）を測定した．

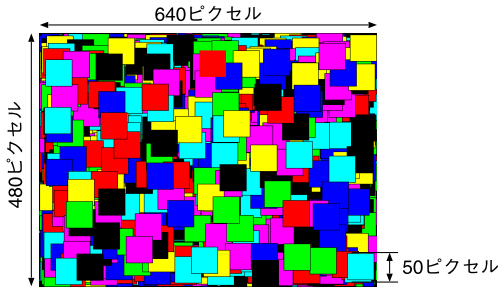


図 10　hdbench clone の描画の例（rectangle）
Fig. 10　Screen-shot of hdbench clone (rectangle).

表 2　更新検出処理時間
Table 2　Update detection time.

hdbench 描画パターン	リニアフレームバッファ	ページタイルフレームバッファ
rectangle	35.4 msec	32.1 msec
circle	42.2 msec	40.2 msec
text	48.8 msec	45.0 msec
scroll	29.1 msec	26.0 msec

更新検出はページテーブル検索方式を用いた．

図 10 に，hdbench clone の描画の一例を示す．hdbench clone は，640 ピクセル × 480 ピクセルのサイズのキャンバスに，50 ピクセル四方の矩形と円形，テキスト，横縞模様のスクロールを高速に描画する動作を行う．アイコンや Web のバナーアニメの描画や，テキスト入力，ウインドウのスクロールと見なせる描画であることと，横長の描画や縦長の描画のように，一方に有利になるような描画でないことから採用した．

更新検出処理時間は，図 9 に示すように，タイマをトリガとして，ページテーブルの更新ビット参照から更新ビットクリアまでの一連の処理に要した時間を測定した．本測定には，ユーザ端末へのネットワークの伝送は含まれていない．描画を 20 秒間行い，途中の 10 秒間を測定区間とした．

測定結果を**表 2** に示す．どの描画パターンにおいても，ページタイルフレームバッファが良い結果を示している．キャンバスの高さである 480 ピクセルから，リニアフレームバッファでは，最悪の場合 480 ページフレームで更新が検出されるが，ページタイルフレームバッファでは，キャンバスがタイルの境界に一致せずに最悪の場合でも，たかだか 336 ページフレームで更新が検出されるため，更新検出ページフレーム数の削減の効果が現れていると考えられる．計測した処理時間には，画像圧縮の時間と送信キューへの追加時間が含まれているため，削減した更新検出ページフレーム数の割合に比例はしていない．この測定では，全般的に処理時間がおおむね 3 msec（10%）短縮されている．

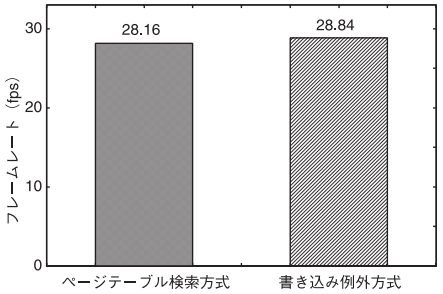


図 11　フレームレート
Fig. 11　Frame rate.

5.2　ページテーブル検索方式と書き込み例外方式の性能比較

次に，ページタイルフレームバッファの採用を前提とした場合に，ページフレームの更新有無の管理方式の比較として，ページテーブル検索方式と書き込み例外方式の間で性能差があるか否かの確認を行った．

評価は，1 台の仮想計算機で Web ブラウザの動画アプリケーションを動作させ，ページタイルフレームバッファを用いた前提で，ページテーブル検索方式と書き込み例外方式の両者について行った．動画アプリケーションは，480 ピクセル × 400 ピクセル で 30 fps の動画を表示する．描画パターンは，全ピクセルを更新描画するものであるため，絵柄の状態を問わず，更新されるタイルの数は一定である．また，更新を検出したフレームバッファ中のページフレームに対しての，現在とバックアップのバイト比較も省略し，絵柄の状態に依存して画像圧縮の処理量が変化する要因を排除することでページテーブル検索と書き込み例外の負荷比較が行えるようにした．

測定は，フレームレート，CPU の使用率，Xen の統計情報を対象とした．**図 11** にフレームレートの結果を示す．フレームレートは，表示端末における表示回数ではなく，1 秒あたりにデバイスエミュレータによる画面の更新検出処理が行われた回数である．また，**図 12** に CPU 使用率を示す．CPU 使用率は，管理ドメインの`/proc/stat` の値を参照し，（全体の時間 − `idle` 時間）を CPU 使用率とした．

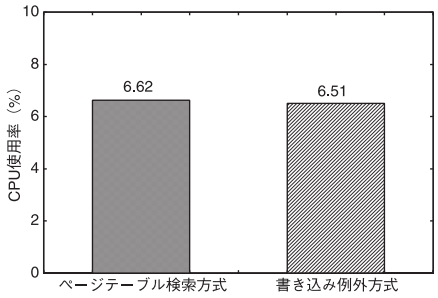


図 12　CPU 使用率
Fig. 12　CPU utilization.

これらの結果によると，ページテーブル検索方式と書き込み例外方式の間には，ほとんど性能差が見られない．一般には，ページテーブルは CPU 内部で保有する機能で更新ビットの設定には負荷はなく，例外のオーバヘッドは大きい．しかし，今回の仮想計算機の環境では，ページテーブルはシャドーページングと呼ばれる MMU 機能の仮想化により実現されており，例外ハンドラによる更新ページフレームの検出処理との間には，負荷による違いはほとんどないことが分かった．

一方，CPU 使用率は双方 6.5%強程度であるが，フレームレートは 30 fps や，30 ms 周期の上限の 33 fps には到達していない．デバイスエミュレータには，画面の変化が少ない場合の負荷低減策として，画面の更新検出の周期の制御が既存の状態で搭載されている．画面の更新検出処理で画面全体に変化がなかったことを検出すると，周期を 50 ms ずつ延ばし，画面の変化を検出すると周期を半分に，また，端末からの入力があると，周期を初期値の 30 ms にしている．このため，30 fps の動画の表示を 30 ms 周期で更新検出処理すると，画面全体に変化なしを検出し，検出処理周期の制御が働くため，28 fps 強になっている．

次に，**表 3** に Xen の統計情報を示す．統計情報は xenperf ツールを用いて取得した．使用した仮想計算機サーバはマルチコア CPU のマルチプロセッサ構成であるため，全コアの結果の和をとった．この結果によると，ページテーブル検索方式に比べ，書き込み例外方式は，VM Exit が 2.7 倍，シャドーページ例外が 9.7 倍，ゲストページテーブルウォークが 3.9 倍に増加している．シャドーページ例外の回数増加については，フレームバッファへの書き込みを例外で検出するためであるが，予想外にゲストページテーブルウォークの回数も増加している．書き込み時の例外処理で行っているページテーブル中の書き込み許可

表 3　Xen 統計情報
Table 3　Statistics information of Xen.

統計情報項目	ページテーブル検索方式	書き込み例外方式
VM Exit	173890	462348
シャドーページ例外	32961	318510
ゲストページテーブルウォーク	99290	385131
シャドーハッシュミス	99	143
シャドーページ割当て	99	143
シャドーページ解放	96	142
シャドーページリサイクル	11	16
validate_gl1e 関数呼び出し	111	58
ゲストページテーブルアクセスビット更新	18	3

ビットの操作が，影響している可能性がある．発生パターンにも依存するが，このように，クリティカルセクションの割合が高くなりがちな処理の回数が増えることは，計算機の処理効率を下げる可能性があると推測できる．また，その他 VM Exit をはじめ，通常の CPU 命令の実行よりもペナルティが高いほとんどの統計が増加傾向にあるため，現時点の測定では図 11，および，図 12 の結果には現れていないが，書き込み例外方式は潜在的に高負荷，あるいは，低効率の可能性が高い．この潜在負荷が，画面更新検出のタイマ周期といったある閾値を超えたときに，フレームレートへも影響しはじめてくるのではないかと考えられる．一方で，現時点の測定においては，統計情報が増加している書き込み例外方式がフレームレートや CPU 使用率の測定結果ではページテーブル検索方式よりも良い結果となっている点から，VMM 内部で適切なタイミングに処理を実行することが，低い CPU 使用率でフレームレートを改善できる可能性を示している．

5.3　実装の汎用性

最後に，試作で行った実装について汎用性という観点で検討する．

ページタイルフレームバッファの実装には，管理ドメインのデバイスエミュレータだけではなく，ゲストドメイン側のグラフィックライブラリに相当する Xorg への改変が必要であった．限定的な部分のみであるが一部改変が必要であった点は，ゲストドメインの汎用性という観点ではマイナス要因である．一方で，グラフィックライブラリによる画面転送対応と比較すると，改変量が限定的で，画面転送プロトコルの統一も維持されており，KVM の統合という視点では依然有利であろうと考えている．ゲストドメインのページタイルフレームバッファへの対応有無について，デバイスエミュレータ側で共存を図るといった対応をとることで，汎用性を維持することも可能ではあるが，グラフィックライブラリへの改変を不

要とする方策は，今後検討しなければならない．

ページテーブル検索方式と書き込み例外方式の実装に関しては，VMM と管理ドメインの改変にとどまっているため，ゲストドメインの汎用性は良好である．

6. おわりに

本稿では，仮想デスクトップ環境における仮想計算機サーバの仮想ディスプレイの処理負荷軽減法について述べた．MMU のメモリ更新検出機能の使用を前提に，ページフレームのサイズと対応したタイルでフレームバッファを構成するページタイルフレームバッファについて述べた．また，メモリ更新検出法として MMU のページテーブルの更新ビットを使用したページテーブル検索方式と，メモリ書き込み例外を使用した書き込み例外方式について述べた．

実装を行い，ビデオカードベンチマークソフトと Web 動画の表示を行った簡易評価で効果を確認した．まず，ページタイルフレームバッファと従来のリニアフレームバッファの画面の更新検出処理時間を比較した．ビデオカードベンチマークソフトの描画実験の範囲内では，ページタイルフレームバッファが処理時間を 10%程削減することを示した．次に，ページテーブル検索方式と書き込み例外方式の比較を行った．フレームレート，CPU 使用率，仮想計算機の統計情報を測定した．測定により，フレームレート，CPU 使用率ではわずかながら書き込み例外方式が良い結果であったが，統計情報から，書き込み例外方式は多くの VMM の処理を実行していた．これらの結果から，仮想ディスプレイの画面の更新検出処理という視点では有意な差はないが，潜在的には書き込み例外方式は高負荷であることが分かった．

今後は，現実のアプリケーションによる評価や，ページテーブル検索方式と書き込み例外方式を組み合わせることでより効果的に画面更新を検出する方式の検討などを行う．また，Extended Page Table（EPT）や Nested Page Tabel（NPT）を用いた仮想計算機環境における方式の検討にも今後取り組みたい．

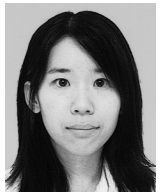
参考文献

1) 後藤真孝，村井信哉，山口健作，田中信吾，西林泰如：ネットワークディスプレイシステムの提案と試作，信学技報，Vol.106, No.577, pp.25–28 (2007).
2) 峰松美佳，後藤真孝，西林泰如，村井信哉：ネットワークディスプレイシステムにおける仮想計算機サーバの試作，2008 年信学総大，D-6-14, (Mar. 2008).
3) 後藤真孝，西林泰如，峰松美佳，村井信哉：仮想計算機環境における画面伝送処理の効率化，2008 年信学総大，D-6-13 (Mar. 2008).
4) 後藤真孝，峰松美佳：仮想計算機における書き込み例外を用いた画面更新検出法の評価，マルチメディア，分散，協調とモバイルシンポジウム（DICOMO2009），pp.1717–1724 (July 2009).
5) Windows Server 2008 Terminal Service. http://technet.microsft.com/ja-jp/windowsserver/
6) Richardson, T., Stafford-Fraser, Q., Wood, K.R. and Hopper, A.: Virtual Network Computing, *IEEE Internet Computing*, Vol.2, No.1, pp.33–38 (1998).
7) Ausbeck, Jr., P.J.: A Streaming Piecewise-Constant Model, *Data Compression Conference* (*DCC '99*), p.208 (1999).
8) Baratto, R.A., Kim, L.N. and Nieh, J.: THINC: A virtual display architecture for thin-client computing, *Proc. 20th ACM symposium on Operationg Systems principles*, October 23-26, Brighton, United Kingdom (2005).
9) Open Graphics Library. http://www.opengl.org/
10) Andres Lager-Cavila, H., Tolia, N., Satyanarayanan, M. and de Lara, E.: VMM-independent graphics accelaration, *Proc. 3rd international conference on Virtual execution environments*, June 13-15, 2007, San Diego, California, USA (2007).
11) Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I. and Warfield, A.: Xen and the art of virtualization, *Proc. 19th ACM symposium on Operationg systems principles*, Oct. 19-22, 2003, Bolton Landing, NY (2003).
12) Huang, X. and Lei, H.: Accelerating the Emulational Device Model in Virtualization System, *2008 International Conference on Embedded Software and Systems*, pp.181–186 (2008).
13) Huang, X.: [PATCH]RFC: VGA accleration using shadow PTE D-bit to construct LFB dirty bitmap, XenSource developer mailing list (July 2007). http://lists.xensource.com/archives/html/xen-devel/2007-07/msg00842.html

(平成 22 年 7 月 26 日受付)
(平成 22 年 11 月 17 日採録)

後藤　真孝（正会員）

平成 9 年九州大学大学院システム情報科学研究科情報工学専攻修士課程修了．同年（株）東芝入社．オペレーティングシステム，通信プロトコルの研究開発に従事．現在，（株）東芝研究開発センター主任研究員．



峰松　美佳

平成 17 年慶應義塾大学大学院政策・メディア研究科修士課程修了．同年（株）東芝入社．研究開発センターネットワークシステムラボラトリーにて，双方向リアルタイム通信処理の研究開発に従事．