

2.6系カーネルに対するLinux Super Page の実装と性能評価

木下 修平[†], 清水 尚彦^{††}

[†] 東海大学大学院 工学研究科 ^{††} 東海大学 情報理工学部

メモリを多く使用するアプリケーションは TLB(Translation Lookaside Buffer) ミスが原因で高速に実行できない場合がある。最近のプロセッサは TLB の有効範囲を拡張するために複数のページサイズをサポートしているが、多くのオペレーティングシステムは固定ページサイズまたは 1 プロセスで 1 つのページサイズのみをサポートしている。我々はユーザプログラムから不可視な形で要求されたメモリサイズに対し、動的に複数のページサイズを割り当てる Super Page を Linux に実装することで、TLB ミス削減によるアプリケーション性能の向上を図る。

An Implementation of Linux Super Page on Linux Kernel 2.6 and Performance Evaluation

Shuheki KINOSHITA[†] Naohiko SHIMIZU^{††}

[†] Graduate School of Engineering, Tokai University

^{††} School of Information Technology and Science, Tokai University

We might not be able to execute applications which use a lot of memory on high speed due to the TLB misses. Recent processor has supported two or more page sizes to expand the TLB coverage. But most operating systems support only the fixed page or only one size in each process. Therefore, we implement Super Page on Linux. Super Page dynamically allocates two or more page sizes for the memory size which demanded from user program. As a result, we reduce the TLB misses and improve the application performance.

1 はじめに

近年、アプリケーションの大規模化やオペレーティングシステムのサポート要求に伴い、従来より増して多くのメモリを要求するようになった。メモリを多く使用するアプリケーションでは TLB のマッピング不足が性能低下を引き起こす場合がある。TLB は仮想アドレスとメモリにあるアドレス変換テーブルを用いて算出した実アドレスの変換結果を蓄えておき、次回からのアドレス変換を高速に行うためのバッファである。アクセスした仮想アドレスに対する適切な実アドレスが TLB エントリにないことを TLB ミスと呼び、これは余分な処理サイクルを発生させる。TLB のエントリを増やすことはプロセッサの設計上困難であるため、近年のプロセッサは複数のページサイズをサポートすることで TLB の有効範囲を拡張している (表 1)。このため、以前より本研究室では Linux Super Page プロジェクトを行ってきた^{1) 2) 3)}。本稿では、今回我々が実装した 2.6 系 Linux Super Page カーネルについて、その設計方針や実装方法について述べる。また Super Page 実装後に行列転置ベンチマークプログラムを用いて行ったオリジナルカーネルとのメモリ転送性能比較を示す。

Table 1 ページサイズ

CPU	Page Size
Alpha	8K, 64K, 512K, 4MB
Sparc64	8K, 64K, 512K, 4MB
IA32	4K, 4MB

2 従来技術

これまでに SGI 社製の IRIX や HP 社製の HP-UX といった商用オペレーティングシステムが複数ページサイズをサポートする Super Page の機構を持つと報告されている。しかし、これらのオペレーティングシステムではユーザが適切なページサイズを設定する必要があるなど問題点がある²⁾。また、最近では 2.6 系 Linux カーネルに HugeTLB ファイルシステムが導入され、プロセッサのサポートするラージページを利用できるようになった。Linux はシステム起動時に HugeTLB ファイルシステム専用のメモリ領域の確保を行い、確保したメモリ領域をフリーリストで管理する。ユーザは疑似ファイルシステムである hugetlbfs を通し

て HugeTLB 機能を利用でき、プロセスがメモリを要求した際、ラージページのフリーリストからメモリを割り当てる。図 1 に HugeTLB ファイルシステムの概念図を示す。

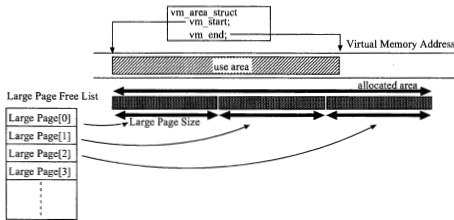


Fig. 1 HugeTLB ファイルシステム概念図

HugeTLB の導入により多くのメモリを利用するアプリケーションの性能向上が期待できるが、以下の問題点も存在する。

- 移植性のないプログラム
特殊な hugetlbfs をマップする必要があり、HugeTLB 用に記述されたプログラム以外は利用することができない。このため HugeTLB ファイルシステムを利用するプログラムに移植性はなく、利用する場合は HugeTLB 用にプログラムを書き直す必要がある。
- スワップ
HugeTLB ファイルシステム上のメモリは常に物理メモリに配置され、スワップ対象外となる。このため、ラージページ用に確保したメモリを使い果たした場合、プロセスはアプリケーションの実行を終了させる。

このように、いくつかのオペレーティングシステムでは、Super Page を利用するための仕組みが提供されているがユーザにとって利便性の良いものではない。このため、我々は実用的で透過性のある Super Page 機構を Linux に実装する。

3 Linux のメモリ管理方法

3.1 仮想メモリ管理

図 2 に Linux における仮想メモリ管理の概要を示す。

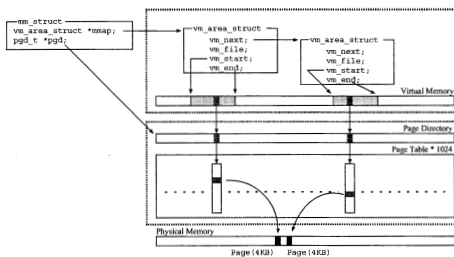


Fig. 2 Linux の仮想メモリ管理

各プロセスは仮想アドレスをマップするために 'mm_struct' 構造体を持つ。'mm_struct' 構造体

は仮想アドレスを実アドレスへ変換するための重要な情報を持つ。そして仮想アドレスは 'mm_struct' 構造体からリンクされている 'vm_area_struct' 構造体によってマップされ、アドレス変換はページディレクトリを用いて行われる。実際の IA32 アーキテクチャにおけるアドレス変換を図 3 に示す。32 ビットの仮想アドレスは PGD、PTE、ページオフセットの三部分に分けられ、物理アドレスを得るために PGD と PTE を用いて 2 段階のページ変換を行う。Linux ではアドレス変換の実装部分を図 4 に示すような 4 段のページ変換テーブルによりモデル化している。このため、IA32 アーキテクチャを扱う場合は変換テーブルの PUD、PMD のエントリ数を 1 とし、さらにその参照先を PGD の参照先とすることで 2 段階のアドレス変換を 4 段階に見せかけている。

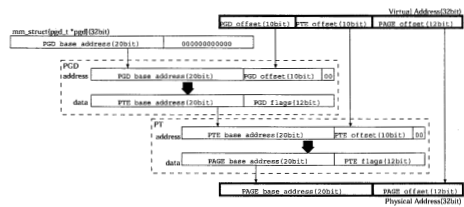


Fig. 3 IA32 の仮想アドレス変換

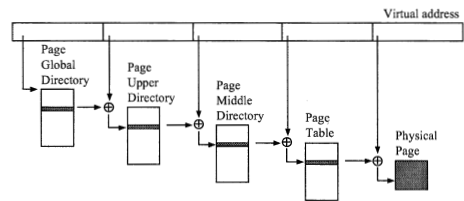


Fig. 4 Linux のページ変換テーブルモデル

3.2 物理メモリ管理

利用可能な実メモリはバディシステムによって空きページとして管理される。バディシステムは空きページを 2 のべき乗単位 (オーダー) で管理し、各オーダーのページ境界に合う場合はより大きなオーダーで管理する。図 5 にバディシステムによるページ管理の概要を示す。

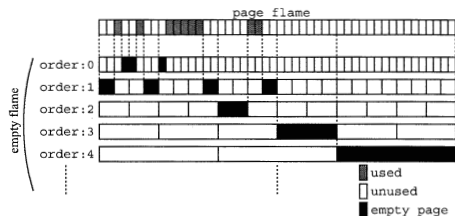


Fig. 5 バディシステムによる空きページ管理

例えばパディシステムが 16 ページ分の連続した空きページの要求を受けると、オーダー 4 のグループに未割り当ての空きページを探しに行く。そして空きページがあれば、その先頭ページのアドレスを返し、ない場合はオーダー 5 のグループから 32 ページを 2 つの連続した 16 ページに分割し、片方の先頭アドレスを返す。このように未割り当ての空きページを見つけるまで順にオーダーの大きいグループへ探しに行く仕組みとなっている。

3.3 メモリ割り当て

ユーザプロセスからのメモリ領域の確保要求が発生するとカーネルは 'vm_area_struct' 構造体を新しく生成し、これを 'mm_struct' 構造体のリンクに追加することでメモリ領域の確保を行う。Linux では不要な物理メモリの使用を防ぐ仕組みを持つため、実際に要求されたメモリ領域に対してアクセスがあるまでは実ページの確保は行わない。このため、アプリケーションが確保要求したメモリ領域へ実際にアクセスするとページフォルト割り込みを発生させ実ページの確保を行う。メモリの開放の際は、メモリ領域を表す 'vm_area_struct' 構造体を 'mm_struct' 構造体のリンクから削除し、そのアドレスに対応する PTE をクリアする。そして、割り当てられていたページが他のプロセスから参照されていないければページを未割り当てリストに追加する。図 6 にメモリ割り当ての流れを示す。

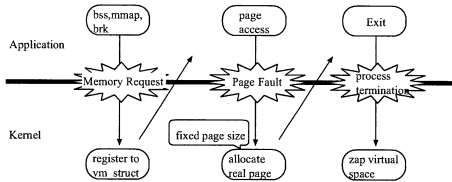


Fig. 6 メモリ割り当ての流れ

4 Super Page カーネル

4.1 設計方針

シンプルで実用的な Super Page Kernel を目指し、参考文献^{1) 2) 3)}に示される基本方針を基礎とした。また今回の設計においては、Super Page の利用効率を向上させるため新たに以下の 2 つの設計方針を基本方針に加える。

- ZAP 時の Super Page ブロックの返却
従来の Super Page カーネルではラージページを開放する際、一度ページサイズをベースページにまでダウングレードさせ、その後パディシステムにベースページ単位での返却を行っていた。この返却方法では、返却途中に他のプロセスにラージページ領域の一部を割り当てられる可能性が生じる。もし返却途中のラージページの一部が他のプロセスにより利用された場合、現在返却中のラージページを次回のラージページ割り当て時に利用することができなくなり、パディシステムの保有

するラージページ数が減少してしまう。実際に 2.5 系 Super Page カーネルを用いてラージページの割り当てと開放を繰り返し、パディシステムが保有するラージページの保有数を調べた (図 7)。

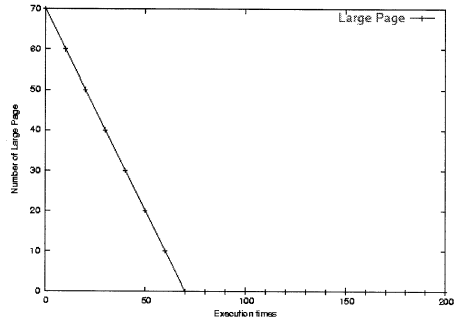


Fig. 7 ラージページ数の推移

この結果から、割り当てと開放の繰り返回数が増加がラージページ保有数の減少に繋がることが明らかとなった。このため、ラージページブロックをまとめてパディシステムに返却することで、上記の問題を解決するとともに返却時間の短縮を図る。

- Super Page 境界へのメモリ割り当て
Super Page を利用するためにはパディシステムとの関係上、要求メモリが Super Page 境界にある必要がある。Super Page 境界にない場合は端数が増加し、ベースページでのメモリ利用が増加する。ベースページの利用は TLB ミスの増加を招くため、端数の増加は性能低下を招くと予想できる。このため、Super Page 境界から Super Page サイズ境界長で可能な限りメモリ割り当てを行い、端数の増加を抑える。

4.2 実装

実装は IA32 アーキテクチャをターゲットとして行い、実装方法は参考文献^{1) 2) 3)}を参考にしている。本 Super Page カーネルの実装方法を説明する。Linux はメモリ管理においてソフトウェアから PT への参照を必要とする場合があるため、4MB 利用時においても 4KB ページでの PT を参照可能にしなければならない。このため、我々はソフトウェア用と TLB 用の 2 つの PGD を確保し、この二つの PGD を使い分けることで Super Page カーネルを実現している。Super Page カーネルのメモリ割り当ての流れを図 8 に示す。ユーザプログラムからメモリ要求された場合、仮想アドレス範囲内の 4MB 境界で、4MB の割り当てが可能である場合、その 4MB 領域の 1024 個の PTE に Super Page 予約フラグを設定する。具体的には PTE のシステムソフトウェア用の空きビットにフラグを立てる (図 9)。これを我々は Super Page 予約と呼ぶ。ただしここ

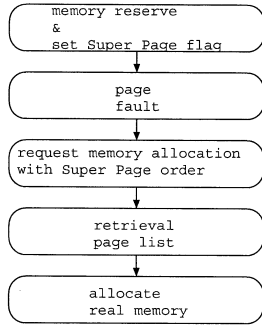


Fig. 8 Super Page カーネルのメモリ割り当ての流れ

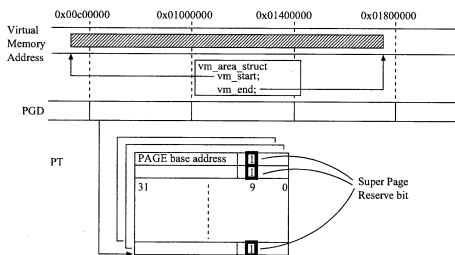


Fig. 9 Super Page 予約

では実メモリの割り当ては行わない。実メモリの割り当てはオリジナルカーネルの方針に従い、ページフォルト割り込み時に行う。割り当て時に Super Page 予約が設定されている場合、パディシステムに対して 4MB の連続したメモリを要求する。4MB メモリの確保に成功した場合、TLB 用の PGD に 4MB ページの割り当てを行う。また、ソフトウェア用の PGD には 4MB ページを 1024 に分け、ページエントリに 4KB ページを割り当てる。4MB の割り当てに失敗した場合は 4KB のベースページでの割り当てを行う。図 10 に 4MB ページにおけるアドレス変換、図 11 に 4KB ページにおけるアドレス変換を示す。4KB ページではソフトウェア用 PGD と TLB 用 PGD は同じ PT への参照値を持ち、4MB ページでは TLB 用 PGD は物理メモリを直接参照可能となる。

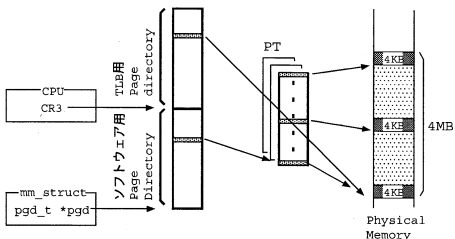


Fig. 10 4MB ページでのアドレス変換

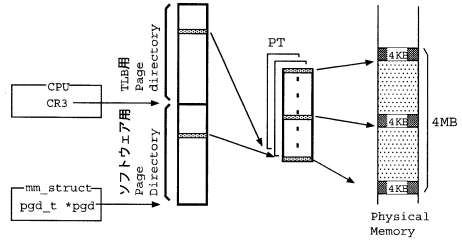


Fig. 11 4KB ページでのアドレス変換

4MB ページを割り当てたページではスワップの発生による一部のページの開放やソフトウェアからの 4KB ページ単位での作業が行われる場合があるため、Super page カーネルにはダウングレードを実装している。ダウングレードは 4MB ページ内の PTE の Super Page 予約フラグをクリアし、ソフトウェア用の PGD を TLB 用の PGD にコピーすることにより行う。また、この際 TLB との整合性をとるため一度 TLB をフラッシュする。

5 評価

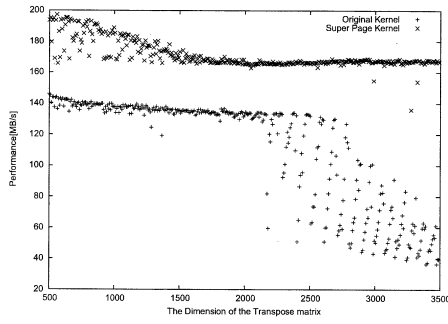
5.1 オリジナル対 Super Page

オリジナルカーネルと Super Page カーネルにおいて図 12 に示す行列転置ベンチマークプログラムを実行し、メモリ転送性能の比較を行った。Super Page カーネルではページアラインを動的に Super Page 境界に合わせるよう実装を行うが、今回の評価時までにはその実装が間に合わなかったため、mmap 関数でのメモリ確保時にユーザが Super Page 境界にアラインを合わせるよう調整を行っている。表 2 は評価を行ったマシン環境であり、以下に評価条件と内容を示す。

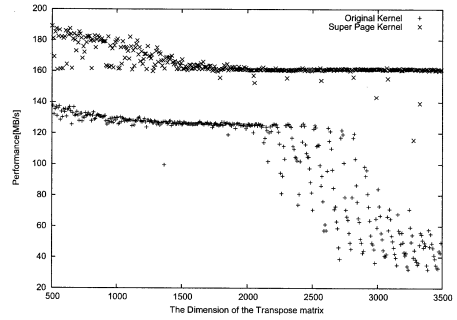
Table 2 評価環境

CPU	AMD Athlon(tm)
CPU Frequency[MHz]	1094.152
Cache Size[KB]	256
Memory [MB]	512
Linux Kernel	2.6.23

- Dimension*Dimension の正方配列 b から同サイズの正方配列 a への転置複製を行う。
- mmap 関数によりメモリを確保する。
- 複製元の配列を縦方向に読み出す方式 (Load Stride) と複製先の配列に縦方向に書き込む方式 (Store Stride) を行う。
- Dimension の値を 500 から 3500 の間で設定する。
- 変更を加えていないオリジナルカーネルと今回実装した Super Page カーネルでプログラムを実行する。

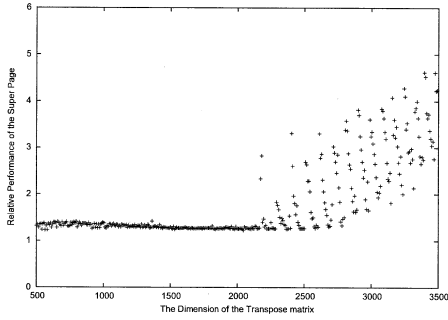


(a) Load Stride のメモリ転送性能

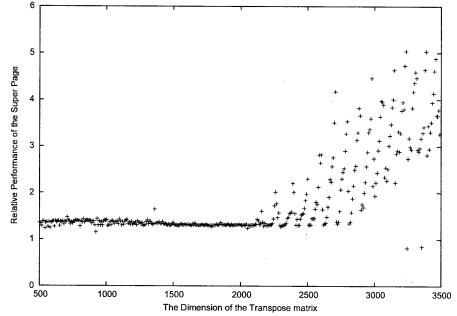


(b) Store Stride のメモリ転送性能

Fig. 13 Load Stride 性能結果

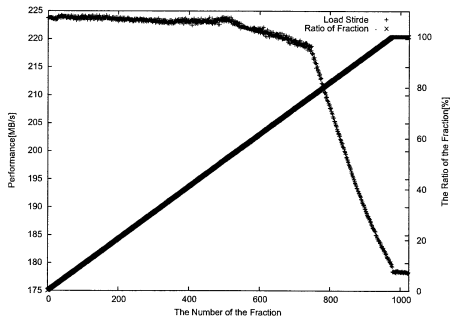


(a) Load Stride のオリジナル対 Super Page 性能比

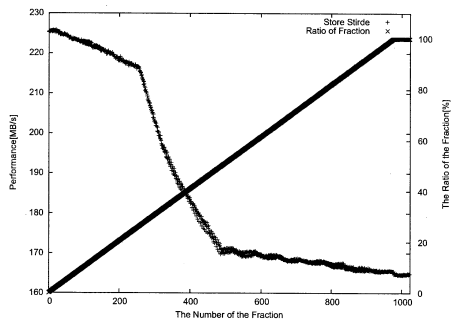


(b) Store Stride のオリジナル対 Super Page 性能比

Fig. 14 オリジナル対 Super Page 性能比



(a) Load Stride



(b) Store Stride

Fig. 15 500 × 500 Matrix における端数増大における性能推移


```

for(k=0; k<ITR; k++) {
    for(i=0; i<dim; i++) {
        for(j=0; j<dim; j++) {
            a[i][j] = b[j][i];
        }
    }
}
}

```

Fig. 12 行列転置ベンチマークプログラム

図 13 にオリジナルカーネルと Super Page カーネルのメモリ転送性能を示す。縦軸は転送性能 [MB/s]、横軸は double 型の正方配列の一边の長さである。図 14 は Super Page カーネルの転送性能をオリジナルカーネルの転送性能で割り求めた性能比である。オリジナルカーネルでは 4KB ページサイズを用いるため、利用メモリが増加は TLB ミスの増加を招き性能を低下させていることが分かる。

5.2 端数増加による性能影響

従来の Super Page では、Super Page 境界に動的にアラインを合わせる実装を行ってきた。しかし、ページアラインを合わせることによる効果についての評価を行っていない。このため、Super Page カーネルにおいて 4MB ページ境界から 4KB 単位で 0 から 1023 までの端数を発生させ、端数増加による性能への影響を調べ、ページアラインの必要性について考察する。図 16 に本評価の概念図を示す。4MB ページ境界からのメモリに対しては 4MB

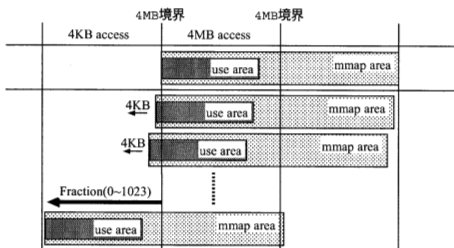


Fig. 16 端数増大における性能評価

ページによるアクセスが可能であり、4MB ページ境界からあふれた端数に対しては 4KB ページでのアクセスとなる。つまり端数の増加は 4KB ページの増加を意味する。図 15 に評価結果を示す。左の縦軸は転送性能 [MB/s]、右の縦軸は端数領域に含まれる配列の割合 [%]、横軸は端数の数である。Store Stride の場合では、端数の増加が性能低下に直接結び付いており、端数の割合が利用領域の約 50[%] に達すると性能は約 50[%] 低下している。これは Stride アクセスする領域から端数となり、TLB ミスを招きやすいことが原因であると考えられる。また、Load Stride の場合では、端数の割合が 80[%] 付近で性能が著しく低下する。これは Stride アクセスする領域が利用メモリの後半部分にあり、端数

の割合が 50 以上 [%] になるまで Super Page 領域に Stride アクセスする領域が存在すること、さらに端数割合が 50[%] 以下のときは端数領域のアクセスは連続したアクセスとなるため、Store Stride アクセスに比べて TLB ミスが発生しにくい状況であることが原因であると考えられる。

Store Stride と Load Stride 実行結果より、端数の増大が性能低下を招くことは明らかである。このため、Super Page カーネルに対して動的にページアラインを合わせる機能を実装することは必要であると言える。

6 まとめ

本稿では、2.6 系 Linux カーネルに実装した Super Page の設計方針や実装方法について述べた。そして、実装後にオリジナルカーネルとの性能比較を行い、Super Page の有効性を示した。また、端数増加による性能への影響を調べ、端数の増加が性能の低下を招くことを明らかにした。これにより、これまでの Linux Super Page プロジェクトで実装してきた、動的にページアラインを合わせる機能の必要性を証明することができた。今後、本 Super Page カーネルにページアライン機能とパディシステムへの Super Page ブロック単位でのメモリ返却の実装を行う。

参考文献

- 1) 高取 研, "IA32 版 Linux Super Page の開発", 2001F 年度 東海大学 工学部 通信工学科 学部論文, 2001
- 2) Naohiko Shimizu, Ken Takatori, "A Linux Super Page Kernel for Alpha, Sparc64 and IA32 -Reducing TLB Misses of Applications", MEDIA 2002 workshop, 2002
- 3) 早坂 晴康, "シャドーページディレクトリを用いた IA-32 版 Linux Super Page の実現と Super Page 端数増大によって起こるメモリ転送性能低下に対する Page Coloring による改善", 2003 年度 東海大学大学院 工学研究科 電気工学専攻 修士論文, 2003
- 4) 高橋 浩和, 小田 逸郎, 山幡 為佐久, "Linux カーネル 2.6 解説室", ソフトバンク クリエイティブ株式会社, 2007
- 5) Daniel P. Bovet, Marco Cesati, "詳解 Linux カーネル 第 3 版", 株式会社オライリー・ジャパン, 2007
- 6) Linux Kernel Hack Japan, <http://hira.main.jp/wiki> (2007 年 12 月 12 日)
- 7) 清水研究室 Linux Super Page -Transparent super page support for Linux, <http://shimizu-lab.dt.u-tokai.ac.jp/lsp.html> (2007 年 12 月 12 日)