システム評価 5－2
（2003．3．7）

Linux Super Page とページカラーリングによるメモリ性能の評価

早坂晴康† 清水尚彦††

今日のプロセッサとメモリの性能格差を埋めるためにレイテンシ隠蔽技術が開発されてきた．キャッシュメモリもその 1 つであるが，CPU のキャッシュはハードウェア処理のため自動で処理されてしまう．そのために 1 プロセスに対し，同一インデックスを割当ててしまうとキャッシュ効率が著しく低下してしまう．本稿では，Linux でインデックスを管理するページカラーリングの性能，Super Page Kernel との統合について報告する．

Evaluation of Memory Performance by Linux Super Page and Page Coloring

Haruyasu Hayasaka† and Naohiko Shimizu††

In order to bury the processor/memory performance gap, latency tolerance technologies has been developed. The cache memory is also one of them. The CPU cache are processed automatically, because of hardware processing. Therefore, to one process, if the same index is assigned, cache efficiency will fall remarkably. In this paper, we report the performance of a page coloring of managing an index by Linux, and integration with Super Page Kernel.

1. はじめに

今日のプロセッサとメモリの性能格差を埋めるためにレイテンシ隠蔽技術が開発されてきた．キャッシュメモリもメモリレイテンシ隠蔽のための 1 つの技術であるが，ハードウェア処理されるためソフトウエアで操作することはできない．そのために 1 プロセスに対し，同一インデックスを割当ててしまうとキャッシュ効率が著しく低下してしまう．そこで効率低下を抑えるために，オペレーティングシステムでインデックスを管理し，メモリ割当ての際にできるだけインデックスの異なるアドレスを渡す Page Coloring という手法がある．本稿では，Linux でインデックスを管理する Page Coloring の方法と性能について報告する．

また，近年は多くのメモリを使うアプリケーションでは TLB ミスも，致命的な障害となっている．TLB ミスに関しては，当研究室で複数の粒度のページを扱うことのできる Linux Super Page[2)3)] を提案してきた．しかし，すべての要求メモリを Super Page に割当てることはしないので，端数が残ってしまう場合がある．そこで，Page Coloring がその端数に効果があるのではないかと Super Page Kernel と Page Coloring の統合について検討してきた．本稿では，さらに Super Page Kernel との統合についての検討とその性能も報告する．

以下，2 章で Linux のメモリ管理とキャッシュ構造について説明し，3 章で Page Coloring の設計を説明する．4 章で Super Page Kernel と Page Coloring との統合の検討，5 章で性能評価をし，6 章でまとめる．

2. Linux におけるメモリ管理とキャッシュ構造

2.1 メモリ管理のための単位

仮想メモリ管理ではメモリをページという単位で管理している．これは，実装されている物理メモリをプロセッサごとに決まっている容量（例えば，IA32 で 4kByte，alpha で 8kByte など）で区分けし，1 つ 1 つの独立した領域として扱う構造である．この区分けされた領域をページと呼び，ページを単位としてメモリの確保，開放などの管理を行っている．

2.2 Buddy Sytem による物理メモリの管理

Linux の物理メモリ管理方式は Buddy System と呼ばれている．これは物理的に連続するページを複数のグループに分けて管理する方式である．Linux2.4 系でのメモリ管理を説明するために，ファイル mmzone.h

† 東海大学 大学院 工学研究科
Graduate school of Engineering, Tokai University
†† 東海大学 電子情報学部 コミュニケーション工学科
School of Information Technology and Electronics, Tokai University

の抜粋を図 1 に示す．

include/linux/mmzone.h

```
#ifndef CONFIG_FORCE_MAX_ZONEORDER
#define MAX_ORDER 10
#else
#define MAX_ORDER CONFIG_FORCE_MAX_ZONEORDER
#endif

typedef struct free_area_struct {
        struct list_head        free_list;
        unsigned long           *map;
} free_area_t;
              ⋮
typedef struct zone_struct {
                              ⋮
    free_area_t       free_area[MAX_ORDER];
                              ⋮
} zone_t;

#define ZONE_DMA                0
#define ZONE_NORMAL             1
#define ZONE_HIGHMEM            2
#define MAX_NR_ZONES            3
              ⋮
typedef struct zonelist_struct {
        zone_t * zones [MAX_NR_ZONES+1];
                                        // NULL delimited
} zonelist_t;
```

図 1　include/linux/mmzone.h からの抜粋

Linux2.4 系では物理メモリを 3 つの「ゾーン」（zone）に分けて管理している．各ゾーンは，zone_struct 構造体で表され，未使用メモリのリストを管理する free_area_struct 構造の配列を持っている．free_area_struct 構造体 free_area[n] は，ページの状態を管理している mem_map 配列の 2^n ページフレーム（PF）連続した領域の先頭アドレスをリスト化している．free_area_struct 構造体の list_head 構造体 free_list がその先頭を示す．通常，配列は 10（free_area[10]）である．図 2 に Buddy System の例を示す．よって，通常は連続した 1 ページフレームから 512 ページフレーム（IA32 では 2MB）まで管理されていることになる．

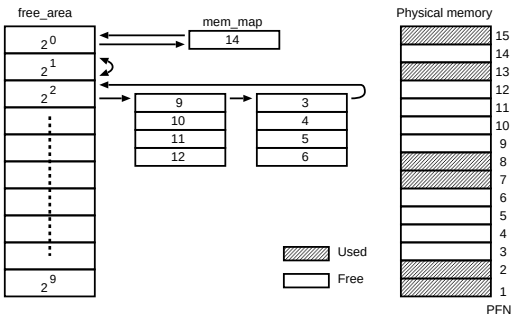


図 2　Buddy System のデータ構造

物理メモリが割当てられていない仮想メモリにアクセスすると，ページフォルトが発生し，alloc_pages() 関数，もしくは__get_free_pages() 関数を希望の連続したオーダーで呼出す．希望のオーダーで未使用の物理メモリがあれば，そのオーダーのリストから削除し，そのアドレスを与える．ないのであれば，1 つ大きいオーダーで割当てを行う．

2.3　キャッシュメモリ

キャッシュはセットアソシエイティブ方式をとっている．メモリにあるアドレスは図 3 に示すように，バイト・オフセット，ブロック内オフセット，インデックス，タグというように分けることができる．インデックスに基づいて，対応するキャッシュのブロックが選択される．そして，アドレスのタグとキャッシュのタグが比較され，そのキャッシュ・ブロックが要求されたものであるかどうかが判断される．さらに，ブロック中の 1 ワードがブロック内オフセットによって選択される．さらに，ブロックを way 数持っているので，同一インデックスのブロックは way 数だけキャッシュ内に存在することができる．

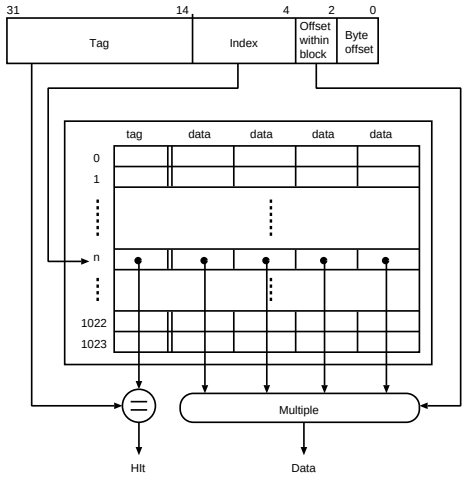


図 3　キャッシュ構造の例

メモリ参照の際に，同一インデックスのブロックに空きがない場合，ブロックを追い出すことになる．

Page Coloring は，物理メモリアドレスのインデックスに相当する部分を分け（Coloring）し，1 プロセスにおいて物理メモリの要求が行われる度に，インデックス（Color）の異なる物理メモリを渡すことによって，キャッシュを効率的に利用するアプローチである．

3.　Page Coloring

今回，Jason Papadouls の Page Coloring カーネル

パッチ[1] を一部修正して Linux-2.4.19 に移植した．

3.1 Page Coloring の導入

Linux では，2.2 に示したようにキャッシュのインデックスを意識した作りにはなっていない．そこで，未使用メモリリストを Color 数用意し，メモリの割当ての際にできるだけ，同一 Color を渡さないように変更する．Color 数は，アーキテクチャによってタグとインデックスのビット数が異なるため，ページカラーリングモジュールを組み込む時に指定するキャッシュ容量をページサイズで割ったものを Color 数としている．例を図 4 に示す．

図 4 Color 数の決定の例

Page Coloring をモジュールにしてカーネルをコンパイルし，Page Coloring モジュールを組み込むと，free_list からつながる未使用メモリのリストを計算した Color ごとに free_area_struct 内に用意した color_list（実際には color_list+Color）に追加する．さらに追加ごとにリストのカウントを行う．図 5 にその様子を示す．

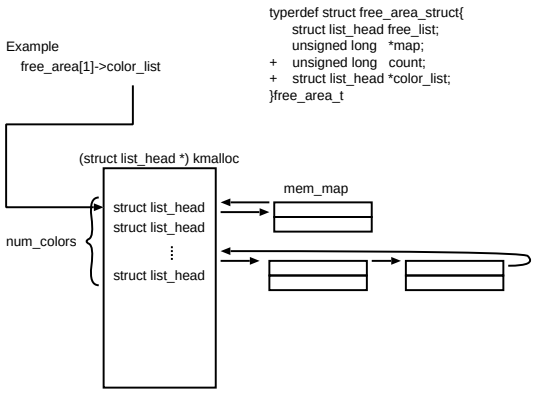


図 5 カラーリスト

リストの Coloring を各ゾーン，各オーダーで行い Page Coloring フラグを立て，システムでの Page Coloring を有効にする．

3.2 rmqueue() 関数

ページフォルトが発生し物理メモリを要求するときには必ず rmqueue() 関数を呼ぶようになっている．通常の Kernel であれは，rmqueue() 関数内で要求ゾーン，要求オーダーの free_area_struct 構造体の free_list から，未使用メモリを探しだすが，Page Coloring フラグが立っているならば，color_list から探す関数 alloc_pages_by_color() を呼ぶようにしている．

3.3 alloc_pages_by_color() 関数

rmqueue() 関数から呼ばれ，そのプロセスで最初の要求だった場合，プロセスにランダムな Color を与える．その Color は，プロセスが持つ task_struct 構造体にあらかじめ用意した場所（target_color）に格納する．これ以後は，この Color を使いメモリの割当てを行う．

要求されたゾーンの要求されたページのオーダー数以上の color_list のカウント数を調べ，カウント（要求されたオーダー以上での未使用メモリのリスト）があるオーダーでの割当てを試みる．カウントのあるオーダーでプロセスに割当てられた Color の未使用メモリがあるのならば，color_list から削除しそのアドレスを返す．もし，そのオーダーでのプロセスに割当てられた Color の未使用メモリがなければ，割当てられた Color を左シフトし，通常と同じように一つ上のオーダーでの割当てを試みる．割当てられた Color すべてのオーダーで割当てが失敗するならば，カラーに 1 を足して同じようにメモリ割当てを試みる．カウント数を調べているので必ずメモリ割当てを行うことができるようになっている．メモリ割当てが成功したら，task_struct 構造体の target_color に 1 を足し，次のメモリ割当てには，Color の異なるメモリを割当てることにする．

4. Super Page Kernel との統合についての検討

通常の物理メモリ割当ては必ず rmqueue() 関数を通して行われ，その関数内で free_area に要求されたオーダーでのページフレームの存在の確認と，存在するのであればリストからページフレームの削除をし，メモリの割当てを行っている．Page Coloring では，requeue() 関数内で Page Coloring フラグを確認し，フラグが立っていれば color_list からページフレームを取り出す関数を呼び出すようにしている．

一方，Super Page はページ・テーブル・エントリ内の Super Page フラグを確認し，フラグが立っていれば Super Page オーダーで，alloc_pages() 関数を呼

び出す．alloc_pages() 関数は，最終的に requeue() 関数を Super Page オーダーで呼出す．

よって Super Page と Page Coloring は完全に仕事の切り分けができるので統合の実現が可能である．まとめると，図 6 のようになる．

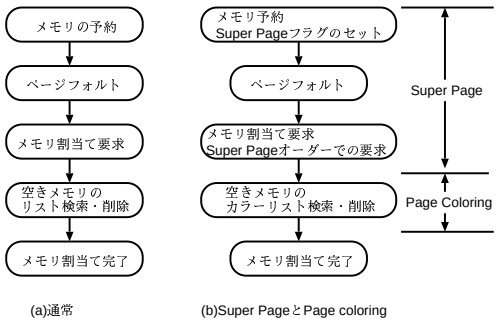


図 6 メモリ割当てまでの流れ

5. 性能評価

行列の転置を行うプログラムで，通常の Kernel，Super Page Kernel，Super Page Kernel+Page Coloring でのメモリ性能の比較を行った．図 7 が行列転置ベンチマークプログラムのロードストライドのコードである．

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

図 7 行列転置ベンチマークプログラム

ベンチマークは $Dimension \times Dimension$ の正方行列 a，b B の転置複製を行っている．今回，Celeron-900MHz/256MB と Pentium4-1.6GHz/1GB でこのベンチマークの実行測定を行った．それぞれのキャッシュサイズは表 1 のようになっている．

表 1 キャッシュサイズ

	Celeron 900MHz	Pentium4 1.6GHz
cache size	128kByte	512kByte

プログラムは gcc コンパイラで”-funroll-loops”オプションを付けてコンパイルを行った．出来るだけプロセッサのメモリパファーマンスのみをテストするために，このコンパイラオプションを指定した．

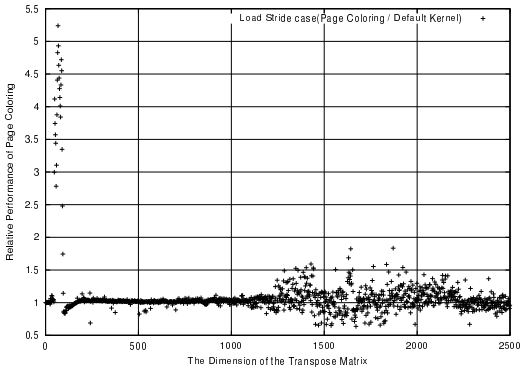


図 8 Celeron での通常の Kernel と Page Coloring とのメモリ転送性能比

図 8 は Celeron での通常の Kernel と Page Coloring を用いた Kernel とのメモリ転送性能比（Page Coloring の性能を通常の Kernel で割ったもの）である．Dimension100 付近で性能の改善が見られる．

図 9 は図 8 の (a) は改善が見られた範囲にしたもの，(b) 比率 0.8 から 1.2 にしたものである．表 2 はそのときのベンチマークの結果の一部であるが，だいたい，Dimension が 89 付近まで性能の改善を見ることができる．これは，

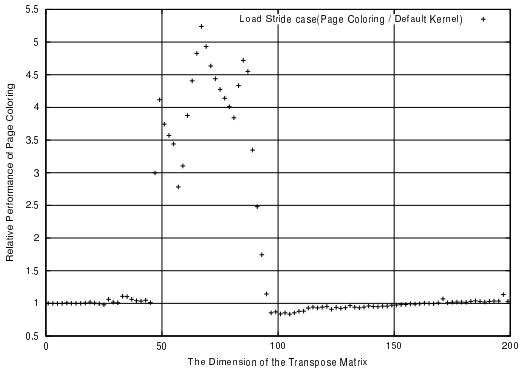
$$Dimension^2 \times 2 \times double = 89^2 \times 2 \times 8Byte = 123.8kByte$$

となり，Celeron のキャッシュサイズのほぼ上限であり，効率的に使われていることが分かる．(b) からは，キャッシュサイズ以上のところでは少しの性能低下が見られるがそれから少しの間，数パーセントの改善が見られる．

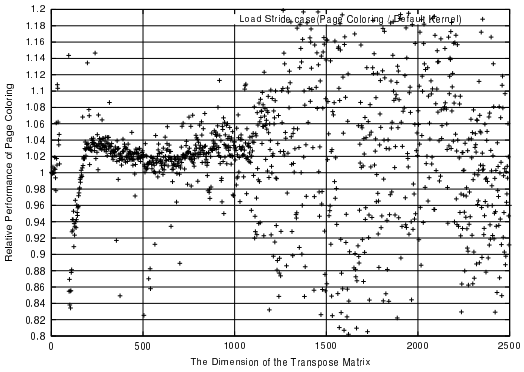
表 2 Celeron のメモリ転送性能（一部抜粋）

Dimension	通常 [MB/s]	Page Coloring[MB/s]
43	1468.12	1537.00
45	1512.98	1529.26
47	519.32	1556.19
49	378.77	1559.53
·····		
85	342.09	1614.42
87	353.40	1608.25
89	361.71	1210.74
91	284.64	705.87
93	261.84	456.61
95	263.04	300.83

図 10 は，Celeron での通常の Kernel と Super Page



(a)Dimension1 から 199 まで



(b) 比率 0.8 から 1.2 まで

図 9 Celeron での通常の Kernel と Page Coloring とのメモリ転送性能比

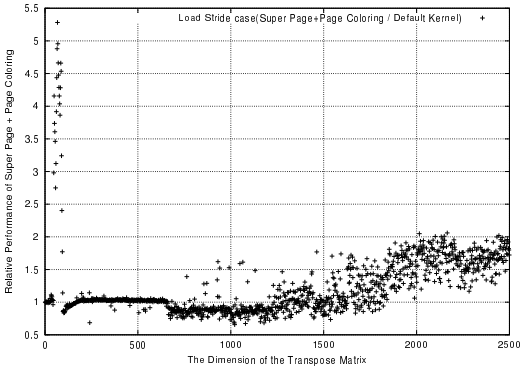
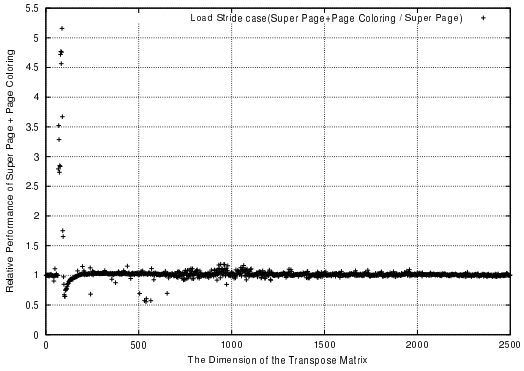
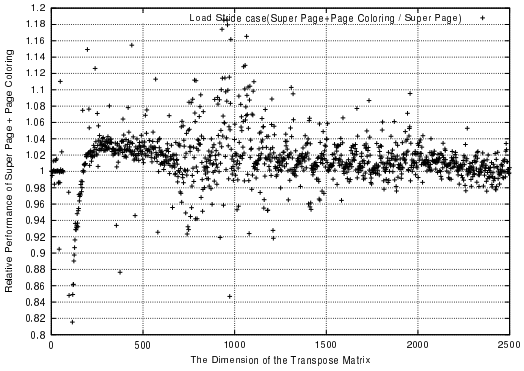


図 10 Celeron での通常の Kernel と Super Page Kernel+Page Coloring とのメモリ転送性能比



(a) 全範囲



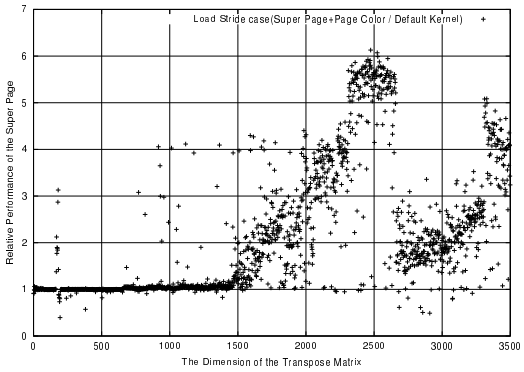
(b) 比率 0.8 から 1.2 まで

図 11 Celeron での Super Page kernel と Super Page Kernel+Page Coloring とのメモリ転送性能比

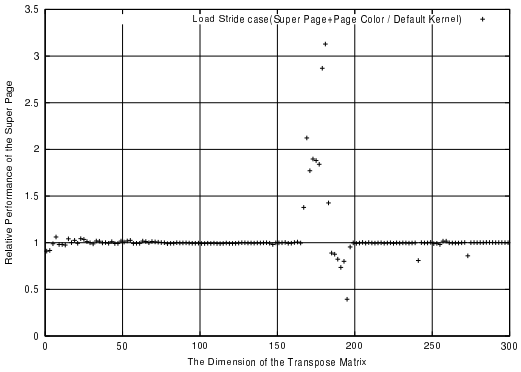
Kernel+Page Coloring とのメモリ転送性能比である．同じく，キャッシュ効率の改善が見られる．

図 11 は，Celeron での Super Page Kernel と Super Page Kernel+ Page Coloring とのメモリ転送性能比である．(a) は測定した全範囲であり，(b) は比率 0.8 から 1.2 に絞ったものである．(b) から，図 9(b) と同じような数パーセントの改善が見ることができる．

図 12 は Pentium4 での通常の Kernel と Super Page Kernel+Page Coloring とのメモリ転送性能比である．(b) から Celeron と同じように，ほぼキャッ

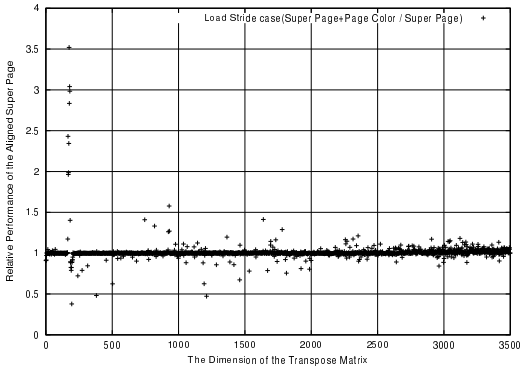


(a) 全範囲

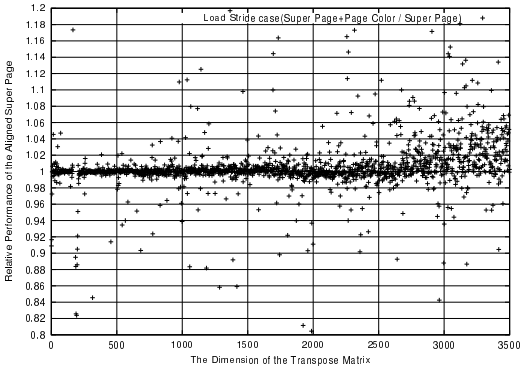


(b)Dimension1 から 299 まで

図 12 Pentium4 での通常の Kernel と Super Page Kernel+Page Coloring とのメモリ転送性能比



(a) 全範囲



(b) 比率 0.8 から 1.2 まで

図 13 Pentium4 での Super Page Kernel と Super Page Kernel+Page Coloring とのメモリ転送性能比

シュ上限である Dimension 181 まで

$$Dimension^2 \times 2 \times double = 181^2 \times 2 \times 8Byte = 511.9kByte$$

の性能の改善が見られる．

図 13 は Pentium4 での Super Page Kernel と Super Page Kernel+Page Coloring とのメモリ転送性能比である．Celeron とは違い大きな Dimension に数パーセントの改善が見られる．

6. ま と め

我々は，Page Coloring によってメモリの物理アドレスを Color 管理することによって，キャッシュ効率の改善を示した．Super Page Kernel との統合が可能なことを示し，Page Coloring を統合しても性能の低下なしに，最大 Celeron で 2 倍，Pentium4 で 6 倍の性能を得た．

今後は，IA32 以外のアーキテクチャでの Page Coloring の効果，転置ベンチマーク以外での性能評価を行っていく．

参 考 文 献

1) Jason Papadouls：http://www.boo.net/ jasonp/page_color-2.4.18-20020705.patch
2) 早坂 晴康，清水 尚彦：“IA32 版 Linux Super Page の実現と評価”, コンピュータシステムシンポジウム 2002
3) Naohiko Shimizu, Ken Takatori：“A Linux Super Page Kernel for Alpha, Sparce64 and IA32 -Reducing TLB Misses of Applications”, MEDEA 2002 workshop
4) 高取 研：“IA32 版 Linux Super Page の開発”, 東海大学 工学部 2001F 年度 卒業研究論文
5) DANIEL P. BOVET, MARCO CESATI：“Linux Kernel”, O’REILLY, 2001
6) ジョン・L・ヘネシー，デイビット・A・パターソン：“コンピュータの構成と設計［下］”，日経 BP 出版センター，1996