# Linuxにおける多粒度TLBのサポートと性能評価

清　水　尚　彦†

HPCにおいてPCクラスマシンを利用することでコストパフォーマンスの良い計算が行えることが知られている。PCクラスマシンの欠点としてメモリ性能においてベクトル機に劣ることが上げられているが、ある分野のアプリケーションではこの原因の一つとして仮想記憶アクセスを高速化するためのバッファであるTLB(Translation Lookaside Buffer)のマッピング能力不足が上げられる。本報告ではプロセッサに備えられる多粒度TLB機構をデマンドページ型仮想記憶を採用するオペレーティングシステムであるLinuxでサポートすることでHPCアプリケーションの性能が向上できることを示す。

# Multigranularity TLB Support for Linux Kernel and the Performance Evaluation

Naohiko Shimizu†

For HPC applications, it is well known that utilization of the PC class machine is the key for a good price per performance. Usually it is pointed out that one of the drawback of PC class machine is the poor memory transfer performance. In some applications, the reason of the poor memory performance is comes from the lack of memory mapping capability of the TLB (Translation Lookaside Buffer). In this report, we will discuss about the performance improvement for HPC applications on a demand paging operating system such as Linux with supporting multi-granularity TLB.

## 1. はじめに

PCクラスマシンの性能は年々大幅に向上しており、特にそのコストパフォーマンスの向上には目を見張るものがある。そのため、HPC分野においてPCクラスマシンを利用することでコストパフォーマンスの良い計算を実現することが一般的になりつつある。PCクラスマシンの欠点としてメモリ性能においてベクトル機に劣ることが知られている。これは一般にメモリとプロセッサの間のバスやメモリのバンク構成などの問題と認識されているが、ある分野のアプリケーションでは仮想記憶アクセスを高速化するためのバッファであるTLBのマッピング能力不足が性能劣化の主因となっている場合がある。これを回避するために特別なブロッキングを用いた演算ライブラリを利用することも多い。しかしながら、本質的にメモリスループットを改善するキャッシュメモリと異なり、TLBは本来OSの管理の容易のために導入されているものである。そこで、TLBによる大幅な性能劣化が起きるシステムは本質的でないプログラムの変更をアプリケーションに求めることになりシステム構成として妥当性を欠いていると言わざるを得ない。

後述するようにTLBミスの影響は大きいがこの傾向はTLBミス時の処理をソフトウェアによって行うRISCプロセッサにおいてより顕著に見られるものである。TLBミスの影響を低減させるためには一つには仮想記憶を用いずに実記憶でプログラムを管理する方法が考えられるが、利用者の立場からみると現実的でない。そこで、プロセッサの機能として1ページを越えるメモリブロックを管理する粒度可変型のTLBを有するプロセッサが存在する[6)7)]。しかしながらいずれのプロセッサもいずれもデマンドページ型のメモリ管理を行うオペレーティングシステムにおいて十分利用されてきたとは言いがたい。[2)]ところが、近年、HPC分野に注力する商用ベンダが性能向上のためこの技術に注目することになりSGI社、HP社がプロセスを指定したページサイズ可変機構のサポートを発表している[3)4)]。

本報告はまず従来のオペレーティングシステムにおいてTLBのマッピング能力の不足で大きく性能が低下することを単純な例題で示し、次にデマンドページ型の仮想記憶管理を用いているオペレーティングシステムにおいてプロセッサに備えられる多粒度TLB機構をサポートする手法を示す。さらにベンチマーク結果によりHPCアプリケーションの性能が向上できることを示す。本報告の実験は主としてTLBの粒度可変機構が全TLBエントリで扱えるCOMPAQ社のAlphaプロセッサを用い、オペレーティングシステムとして著者の馴染みのあるLinuxを用いて行った。

† 清水 尚彦
〒 259-1292 平塚市北金目 1117
東海大学 電子情報学部
TEL: (0463)58-1211　　FAX: (0463)58-8320
email: nshimizu@keyaki.cc.u-tokai.ac.jp

```
program benchmark
parameter (nmax=3001,nloop=500)
parameter (repmax=100000)
double precision a(nmax,nmax)
double precision seconds, s
real dummy(2)
integer i,j,rep
do i=1,nloop
  do j=1,nloop
     a(i,j) = i+j*nmax
  enddo
enddo
s=0
seconds=dtime(dummy)
do rep=1,repmax
  do j=1,nloop
     s = s + a(j,1)
  enddo
enddo
seconds = dtime(dummy)
write(*,*) 'continuous', seconds, s
s=0
do rep=1,repmax
  do i=1,nloop
     s = s + a(1,i)
  enddo
enddo
seconds = dtime(dummy)
write(*,*) 'stride', seconds, s
stop
end
```

**図 1** 行方向および列方向のデータアクセス時間推定のベンチマークプログラム

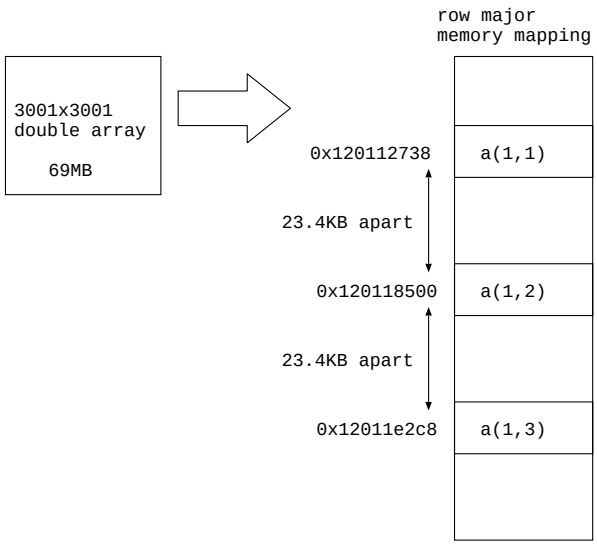


**図 2** 配列のメモリマッピング

## 2. 簡単な例題による性能テスト

図 1 のプログラムは倍精度の浮動小数点の配列のある 1 行もしくは 1 列の 500 要素のデータの和を`repmax` 回繰り返した結果の時間を表示するプログラムである。このプログラムは性能の検証のための合成ベンチマークである。FORTRAN では配列は列方向にたいしてメモリ上連続に配置され、行方向に対してのアクセスは図 2 に示すようにストライドアクセスとなる。

本ベンチマークでは行方向、列方向にそれぞれ 500 要素を読み出している。列方向の読み出しは連続アクセスとなるため、$500 \times sizeof(double) < 4KB$ のデータを扱うにすぎない。これに対して行方向のアクセスではプロセッサがキャッシュライン単位にメモリにデータを移動するため実効的なデータ移動量はさらに大きくなる。しかしながら、キャッシュのブロックサイズが 32 バイトの場合、本ベンチマークはストライドアクセス時でも実効ワーキングセットは $500 \times 32 < 16KB$ となり、多くのプロセッサの一時キャッシュに 十分収まる。したがって、このプログラムで生じた性能の差はキャッシュ性能ではなく、純粋にメモリのアドレス変換オーバヘッドが観測されていると考えてよい。

最適化コンパイラはこのプログラムのループを判断して行列の配置を変更したり、不要なループを削除するので、プログラムのコンパイル時には最適化オプションを最低限に収める必要があることに注意していただきたい。

Linux/Alpha を搭載した Alpha EV67(600MHz) と Solaris を搭載した UltraSparc(400MHz) ならびに Pentium-4(1.7GHz) での実行結果を表 1 に示す。(ここではマシンのベンチマークをするわけではないので、それぞれの性能についてはあえて言及しない。)

**表 1** 連続アクセスとストライドアクセス時のプログラム実行時間

| Access pattern | Alpha EV67 | UltraSparc | Pentium-4 |
|---|---|---|---|
| continuous | 0.303 | 1.763 | 0.15 |
| stride | 6.365 | 7.583 | 0.79 |

表 1 で示すように連続アクセスの場合とストライドアクセスの場合で実行性能に大きな差が見られる。前述したようにベンチマークはプロセッサの 1 次キャッシュに収まる程度のワーキグセット (メモリフットプリント) しか使わないように設計したため、この差は主として TLB マッピング能力の不足によっ

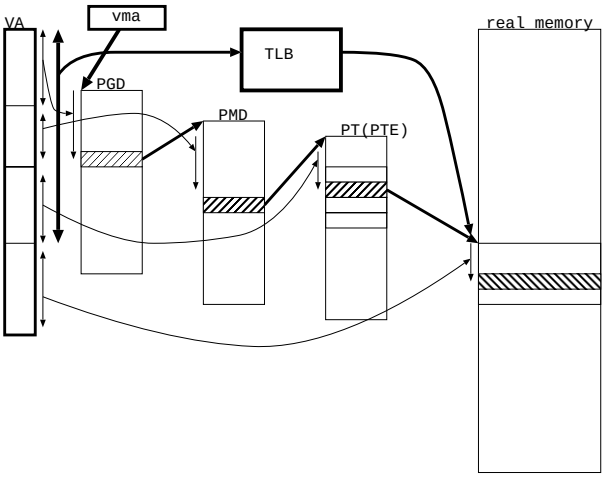


**図 3** Linux における仮想記憶マッピング

て発生したと言える。場合によっては 10 倍以上もの性能差が生じることが分かる。

TLB マッピング能力の不足は HPC(High Performance Computing) 分野だけの問題ではなく、多くのメモリを使用するアプリケーションにとって深刻な問題となる可能性を秘めている。特に、ループ構造が単純であることの多い HPC 分野ではコンパイラがソース構造を解析することで比較的簡単にループ構造の自動変換が実現でき、問題に対処しやすいのに対して、データベースなどのデータ依存の大規模なポインタ構造を用いるシステムではコンパイラがあらかじめ予測してループ変換などのテクニックを適用する余地はほとんどなく、性能上のネックとなる可能性がある。

コンパック社でも OpenVMS においてオラクル社データベース専用に TLB 粒度変更するシステムジェネレーションレベルのメモリオプションを用意しているのもこれらを裏付けるものであろう。

## 3. Linux におけるデマンドページ型仮想記憶管理方式の概要

Linux は従来の汎用オペレーティングシステムと同等のデマンドページ型仮想記憶管理を採用している。図 3 に Linux における仮想記憶マッピングのイメージを示す。仮想アドレス VA は PGD,PMD,PT の 3 段階の変換テーブルを通って実アドレスに変換される。

その一方、3 段階の変換テーブルをすべてのメモリアクセスに適用すると著しい性能ペナルティとなるため、仮想アドレスからダイレクトに実アドレスに変換する変換バッファ(TLB)がプロセッサ内に設置され OS の管理により高速なアドレス変換が実現される。一般に連想メモリで構成される TLB のエントリ数はむやみに多くすることは困難であるため、多くのプロセッサでは 100 個以内の比較的少数の TLB エントリを有するにすぎない。

プロセスの記憶領域は次のように管理される。

**プロセスに割り当てられる仮想空間** プロセステーブル (およびテーブルからリンクされる構造体) に割当が行われる仮想空間の大きさとページテーブルのディスクリプタへの情報が格納される。仮想空間はプロセスを起動するときに親プロセスの空間をコピーすることにより、もしくはロードモジュールをロードする場合にヘッダ情報から静的に割り当てる空間、およびプロセスが動的にメモリ要求を出すことにより割当が行われる場合がある。割当はプロセステーブルに登録される仮想空間の情報を更新するが、プロセスのランタイムの仮想空間を定義するページテーブル自体には変更を加えない。これはページテーブル自身のメモリフットプリントを節約するための処置である。プロセスが実際にメモリを参照するまではページ割当を行わないデマンドページを採用しているが、ページ自身だけでなくページテーブルエントリもデマンドページの対象にしていると考えると分かりやすい。ただし、ページテーブルは 1 エントリ毎には制御できない (ページテーブルの保護もページ単位に行う)。

**ページテーブルメンテナンス** プロセスが割り当てられたメモリをアクセスした場合には変更されていない (すなわち無効となっている) ページテーブルによってプロセッサはページ不在の例外を発生する。ページ不在例外の例外ハンドラにおいてプロセスの仮想空間の範囲にアクセスしたメモリ領域が入っているか否かを検査する。ここで、割り当てられていないメモリをアクセスした場合にはセグメント例外をプロセスに発生させるが、アクセスした領域がすでにプロセスに割り当て済の領域であれば当該仮想アドレスに対応する実ページを割り当てた上でページテーブルを書き換えプロセスの再起動を行う。

**実ページの開放** 空きメモリが少ない場合には新たなメモリを割り当てる前に従来割り当て済のメモリ領域の開放を試みる。アクセス頻度が少ないメモリページがあった場合書き込みされている場合には当該ページをスワップ領域に退避しページテーブルの該当エントリを無効化する。Linux ではページテーブルのページフレーム番号のフィールドにスワップエントリ番号を書き込むことでスワップの管理を容易にしている。

**仮想空間の開放** プロセスが終了する場合もしくは明示的に仮想空間の開放が指示される場合ページテーブルをクリアするとともに他のプロセスとの共有がされていない実ページはフリーページのリストに追加される。

ここに示したように Linux をはじめとするデマンドページ型のメモリ管理を行うオペレーティングシステムはプロセスが実際にメモリをアクセスした時にはじめてメモリの割当を行うようにすることで実メモリの不要な占有を避けている。そ

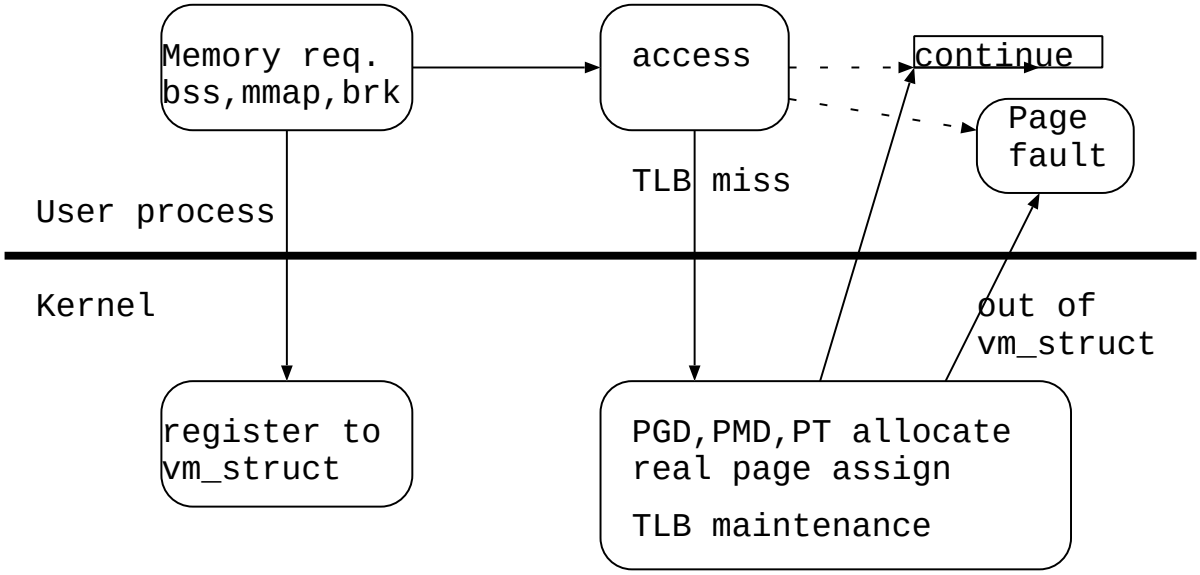


**図 4** メモリ要求から割り当てまで

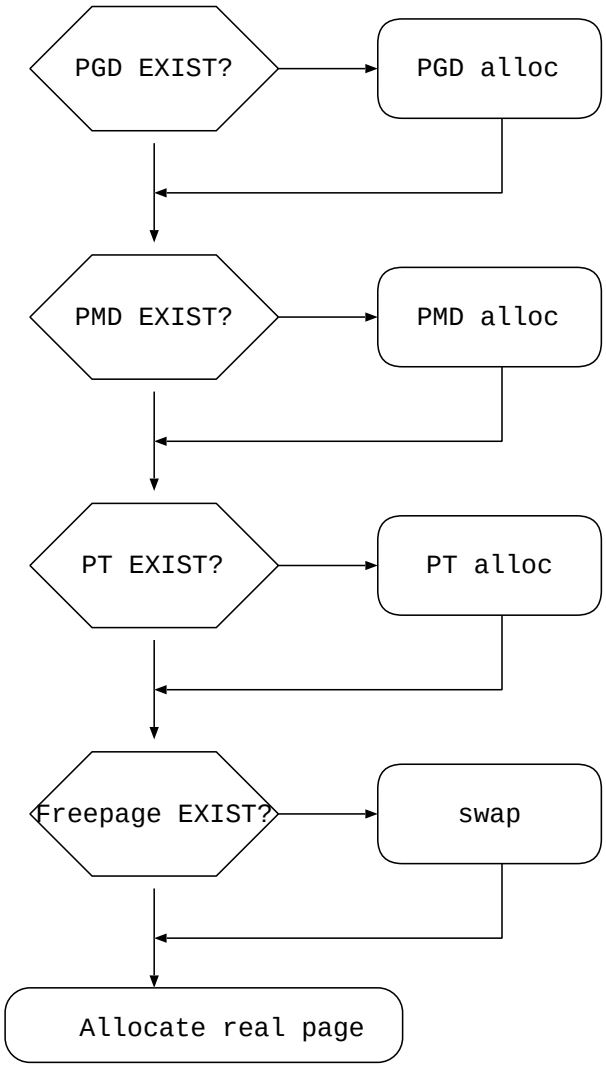


**図 5** メモリ割り当て

こで実際にメモリアクセスがどのアドレスで発生するかをオペレーティングシステムがあらかじめ把握することは難しい。ところが、粒度可変型の TLB を有するプロセッサの多くは指定された粒度の境界に合致した実メモリと仮想メモリの割当を必要とするため従来のページ管理方式では対応できなかった。

## 4. Linux における可変ページサイズサポート案

SUN による検討結果[2] を見ても、カーネルで実際に複数のページサイズを扱うのは極めて困難である。これは UNIX 型のデマンドページ制御ではページサイズを単位として制御する構造が多く存在するためでもある。そこで、SGI、HP 等はページサイズ自身は固定して複数のページをまとめて TLB の 1 エントリにマッピングする方法で疑似可変ページサイズをサポートすることにした。[3][4] 今回検討を行った Alpha プロセッサは後述するように粒度ヒントをページテーブルに設定することでページサイズ自身ではなくページをグループとして扱うことを宣言する機構を有している。そこで、ページグループを扱う形での Linux 可変ページサイズサポートの方針を検討する。

**可変ページサイズでのメモリ割当システムコール新設** 可変ページサイズをサポートしたときに他のプロセスとの競合を一切関知しなくてよいのであれば実現は簡単である。そこで、システムコールの新設によってメモリを固定的に割り付ける本方式が考えられる。もっとも簡単にはメモリを固定する `mlock` のインターフェイスを用いて、ページが主記憶に固定されるときのみ可変ページサイズを有効とするものである。この方式では `mlock` を発行できるスーパーユーザーだけの利用に限定されるが、安全性の高い方法である。

**仮想記憶に可変ページサイズサポートフラグ新設** 可変ページサイズのサポートが必要なアプリケーションの仮想記

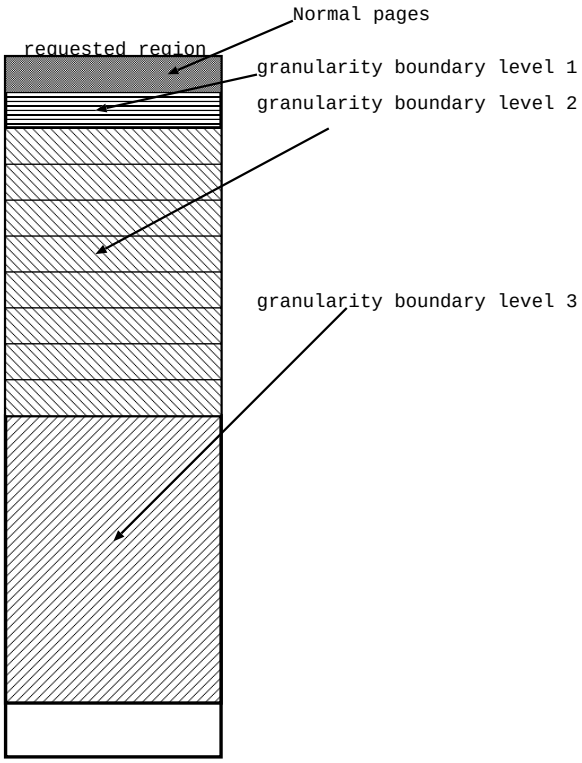


**図 6** 粒度可変ページ割当てと粒度境界

憶システムに特殊なフラグを新設し、このフラグがオンの場合に可変ページサイズを有効にする。この方式ではあらかじめ想定したユーザーのみが可変ページサイズを利用することになるため、不要なオーバーヘッドが必要とならない可能性が高い。[3] メモリ固定は指定しなくても仮想記憶のフラグを判定してカーネルがスワップを抑止するなどの対策を立てやすい。反面、フラグを立てるために特殊なシステムコールなどが必要となるのでバイナリ互換性は低い。SGI ではバイナリをこの方法で動作させるためフラグを立てる特殊コマンドの配下で動作させている。

**オンデマンド可変ページサイズサポート** 可変ページサイズが使用可能な場所ではすべて有効にする。多くのアプリケーションは頻繁にページの状況を変更するのでそれに伴い、可変ページサイズの使用できなくなることも多いため、ページサイズダウングレードの機構の実装が必須である。

## 5. 粒度可変デマンドページ型仮想記憶管理

HPC 分野の計算に使われることの多い Alpha プロセッサの多粒度 TLB の仕組みを用いた粒度可変デマンドページ型仮想記憶管理方式を開発した。

Alpha プロセッサの多粒度 TLB は次のような特徴を有する。

- 1,8,64,512 の各ページ数に対応する粒度をサポートする。粒度情報はページテーブルエントリに記述するが、同一ブロックに属するページに対応するページテーブルエントリは全て同じ粒度情報を持つ。
- 全 TLB エントリがいずれの粒度にも対応できる。粒度毎に使えるエントリ数が限られているプロセッサでは TLB エントリ数をオペレーティングシステムが管理する必要になるので、各 TLB エントリが均質な Alpha プロセッサがオペレーティングシステムには便利である。
- 実／仮想アドレスとも粒度の境界へ合わせる必要がある。プロセッサのハードウェア設計上ページサイズの拡大と同等の構成を取っているため本制限がある。
- ページテーブルは粒度に関わらず 1 ページごと必要。Alpha プロセッサでは TLB の入れ換えは PAL コードと呼ばれるファームウェアで行っているが、このファームウェアは粒度情報の有無に関わらず同一の処理を行うため粒度情報なしの場合に対応したページテーブルを用意しておく必要がある。
- 1 つの粒度ブロックに対応する各ページテーブルエントリの保護情報は全て等しくしなければならない。この制限も上記ファームウェアの動作方式から来ている。

この Alpha の多粒度 TLB の機構を用いて粒度可変機構を Linux に導入するにあたり問題となる点とその対策を示す。

**大きな粒度でのメモリ割当が可能か判断困難** Linux ではプロセスが実際にページ不在例外を発生させた場合に渡される情報は仮想アドレスだけであり、そのままでは大きな粒度でメモリを割り当てるべきか否かが判定できない。プロセスがどれだけの仮想空間を要求したかは要求時点すなわち仮想記憶領域を登録する時点でしか判定できないからである。そこで、仮想空間を要求した時点で大きな粒度の割当をしても構わない要求か否かを何らかの手段で記録する必要がある。プロセステーブルの管理情報を変更するだけで実ページやページテーブルには手を付けていない。実ページを実際に割り付けることは不要なメモリ消費につながるため採り入れられないため、最低限の情報としてページテーブルの情報に割当要求の大きさのヒントとなる情報を保持することとした。

**ページ保護情報が変更される場合の処置** オペレーティングシステムはページ単位の管理を行うためページ保護情報は 1 ページずつ変更される。ところが、同一ブロックに属するページテーブルエントリは全て同じ保護情報を持たなくてはならないため問題となる。これに対して粒度の情報が設定されているページの保護情報を変更する場合には同じブロック内のページの粒度を当該ページの粒度が 1 となるまで再帰的に下げてからページ保護情報を変更する。TLB に設定されている情報は保護情報、粒度情報とも一致しなくなるため無効化する。

これらの対策によって Linux への粒度可変機構を組み込むことができ、SPEC ベンチマークを含むさまざまなアプリケー

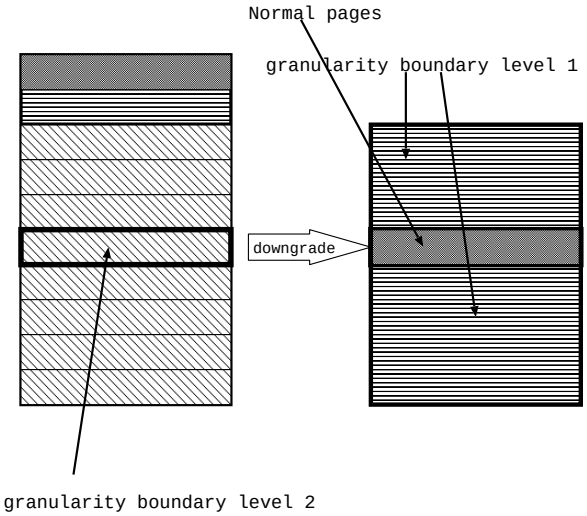


**図 7** 粒度のダウングレード

ションを動作させることができた。

## 6. ベンチマーク結果と考察

**表 2** SPEC CFP2000 ベンチマーク結果数値。Alpha 21264A 667MHz プロセッサで実測。コンパイラは COMPAQ FORTRAN および COMPAQ C を使用。Linux カーネル 2.2.16 と本報告の変更を加えたカーネルを使用。表の値は 300MHz の SUN UltraSPARC プロセッサの値を 100 としたときの相対性能である。

| ベンチマーク名 | Linux オリジナル | 本方法 | 相対性能 |
|---|---|---|---|
| 168.wupwise | 429 | 434 | 1.01 |
| 171.swim | 654 | 727 | 1.11 |
| 172.mgrid | 362 | 361 | 1.00 |
| 173.applu | 393 | 420 | 1.07 |
| 177.mesa | 470 | 487 | 1.04 |
| 178.galgel | 286 | 994 | 3.48 |
| 179.art | 540 | 564 | 1.04 |
| 183.equake | 216 | 214 | 0.99 |
| 187.facerec | 528 | 573 | 1.09 |
| 188.ammp | 341 | 344 | 1.01 |
| 189.lucas | 410 | 513 | 1.25 |
| 191.fma3d | 409 | 381 | 0.93 |
| 200.sixtrack | 273 | 276 | 1.01 |
| 301.apsi | 289 | 497 | 1.72 |
| SPECfp_base2000 | 384 | 452 | 1.18 |

本報告の変更を取り込んだ Linux において SPEC CPU ベンチマークを実行した。CFP2000 の実行結果を表 2 に示す。

ここに示したように多くのアプリケーションにおいて本報告の方法はよいベンチマーク結果を示した。特に 178.galgel では３倍以上もの高速な結果となった。178.galgel は流体力学計算の問題である。同じく大規模にメモリを必要とするアプリケーションにおいても 172.mgrid のようにほとんど効果の出ないアプリケーションも存在する。これは本報告の方法はメモリを連続的にアクセスプログラムに対しては (性能劣化は少ないものの) あまり高速化の効果が期待できない性質がありマルチグリッド手法自身には非連続なメモリアクセスが存在するものの計算量の多いところでは連続アクセスになり計算量の相対的に少ない部分に非連続なアクセスが使われるプログラムの性格によって粒度可変機構の有効性が低減したものと考えられる。

## 7. ま と め

COMPAQ Alpha プロセッサに備わる多粒度 TLB の仕組みを用いて Linux に粒度可変機構の組み込みを行う方法とそのベンチマークによる性能検証を報告した。本報告の手法は数値計算等で良く用いられる SPEC ベンチマークにおいて 17%以上の優れた性能向上効果をもたらした。本手法では既存のアプリケーションに手を加えることなく実行時のメモリ管理を工夫することで高性能が得られるためさまざまな高性能計算の用途に適している。

本報告の方法を組み込んだ Linux はページ粒度の縮小が必要になったときにダウングレード処理を行なうが、この時、TLB の登録データの削除をあわせて行なっている。ところが、実験では TLB の解放に問題があり、ダウングレードを頻繁に行なうアプリケーション (X など) で問題を生じている。この解決は技術的には容易であるが、性能や OS への変更量のトレードオフを現在模索中である。

今後、他プロセッサアーキテクチャへの本方法の移植や本方法を活用できる新しいプロセッサアーキテクチャの提案などを行っていきたい。

## 参 考 文 献

1) N.Shimizu, "Multi-Granularity Page Size Support for Linux and the Performance Evaluation", Wuhan University Journal of Natural Sciences, Vol.6, No.1-2, 2001
2) Y.Khalidi, M.Talluri, M.Nelson, D.Williams, "Virutual Memory Support for Multiple Page Sizes", 4th Int'l Workshop on Workstation Operating Systems, Napa, California, October 1993.
3) N.Ganapathy, C.Shimmel, "General Purpose Operating System Support for Multiple Page Sizes", USENIX1998, Annual Technical Conference. USENIX Assoc., June 1998
4) I.Subramanian, C.Mather, K.Peterson, B.Raghunath, "Implementation of Multiple Pagesize Support in HP-UX", USENIX 1998 Annual Technical Conference. USENIX Assoc., June 1998
5) S.Maxwell, "Linux Core Kernel Commentary", Coriolis, 1999
6) http://www.compaq.com/
7) http://www.intel.com/
8) http://www.spec.org/
9) http://shimizu-lab.et.u-tokai.ac.jp/~enshimizu/