スーパーコンピュータ「京」におけるアプリケーション性能への TLB の影響

黒田明義[†1]　杉崎由典[†2]　千葉修一[†2]　熊畑清[†1]　寺井優晃[†1]
井上俊介[†1]　南一生[†1]

ハイパフォーマンスコンピューティングにおけるアプリケーション性能は，アルゴリズムやシステム特性など様々な要因によって変化する．このためハードウェアの性能を最大限に引き出すためには，ハードウェア，システムソフトウェア，アプリケーション開発者の連携が重要となる．今回，TLB がアプリケーション性能に及ぼす影響について，スーパーコンピュータ「京」上で，いくつかのアプリケーションを用いて解析を行ったので報告する．TLB とは仮想アドレスと物理アドレスの変換を行う機構であるが，どのようなアーキテクチャであれ，キャッシュ同様そのデータアクセスパターンによってアプリケーション性能への影響が予測される．しかし，いつどこで TLB ミス発生するか予測が困難であることから，単純な性能低下ではなく，想定される箇所以外の性能のばらつきとして計測されることが多い．本報では，TLB ミスによって引き起こされる性能への影響について，その原因，特徴，並びに解決方法について，いくつかのアプリケーションの実測結果を元に議論する．

Performance Impact of TLB on the K computer Applications

AKIYOSHI KURODA[†1]　YOSHINORI SUGISAKI[†2]　SHUICHI CHIBA[†2]
KIYOSHI KUMAHATA[†1]　MASAAKI TERAI[†1]　SHUNSUKE INOUE[†1]
KAZUO MINAMI[†1]

Application performance in high-performance computing depends on various factors such as algorithms and systems. To maximize hardware performance, it is important for collaboration among hardware, system software, and application developers. We report the performance impact of TLB on the application by simultaneously running several applications on the K computer. TLB is a mechanism that translates virtual and physical addresses in any architecture. Performance impact of TLB on the application is predicted by the data access pattern, which is similar to that of the cache. It is difficult to analyze when and where a TLB miss may occur, because it is often measured not only as performance degradation but also as performance fluctuation. In this paper, we discuss not only calculation results based on some applications that measure the performance impact by TLB misses but also their causes and characteristics, as well as some ways to resolve them.

1. はじめに

1.1 序章

理化学研究所では，スーパーコンピュータ「京」（以下，「京」と呼ぶ．）の開発と並行して，共用開始時にいち早く「京」での科学的成果を創出すべく，アプリケーションの高性能化を手がけてきた．また 2011 年 4 月からは試験利用として，広くユーザーにもアプリケーションの高性能化，並びに成果創出のために計算資源の提供を行ってきた．それらの過程で見い出された問題点や性能に関する情報は集約して，システム開発にフィードバックを行った．本稿は，その過程で見い出された Translation Lookaside Buffer [1]（以下，TLB と呼ぶ．）によるアプリケーション性能に与える影響について，調べたので報告する．1 節では，背景となる「京」の開発について説明し，2 節では，TLB について，その働きと性能低下の可能性について議論する．3 節では，アプリケーションにおける TLB の性能への影響について，実アプリケーションでの実測をもとにいくつかの例を報告する．4 節では，新規に開発されたコンパイラの最適化機能によって発生し得る TLB ミスについて，実アプリケーションでの実測に基づき議論する．5 節では，ハイパフォーマンスコンピューティングの今後の開発を見据えて考察を行い，6 節にて，TLB によるアプリケーション性能への影響についてまとめる．

1.2 スーパーコンピュータ「京」

「京」とは，理化学研究所計算科学研究機構にて開発されたスーパーコンピュータシステムである．国家基幹技術として，10PFLOPS 級のスーパーコンピュータを開発することで，技術力の維持発展，並びにわが国の共用施設として，幅広い計算科学分野の利用に供する世界最高のコンピューティング基盤を整備することを目的としている．「京」は 2012 年 7 月に完成し，2012 年 9 月からは一般ユーザ向けに共用が開始された．

計算に用いるノードは，1CPU，16GB のメモリ，ノード間のデータ転送を行うインターコネクト用 LSI（ICC: Inter-Connect Controller）で構成されている．CPU は「京」向けに新規で開発された富士通社製の SPARC64™ VIIIfx [2][3][4][5][6]であり，1 CPU 上に 8 コアの演算器を持つ．ピーク性能は 128 GFLOPS で，メモリバンド幅は 64GB/s

†1 理化学研究所
RIKEN
†2 富士通株式会社
FUJITSU, LTD.

(0.5B/F)である．浮動小数点レジスタは 256 本に増強されており，これまでの CPU と比較して，コンパイラによる命令スケジューリングが容易となっている．また SIMD 命令の導入による，ベクトル計算，マスク演算が可能となり，後者はプログラム中の分岐での命令スケジューリングの向上につながる．またセクタキャッシュ機能が導入され，一部再利用性のあるデータを留め置くことが可能となった．

計算に用いるノードは, Tofu インターコネクトと呼ばれる 6 次元メッシュ/トーラスネットワークで結合されている[7][8]．各リンクのバンド幅は 5GB/s(双方向)で，システム全体のバイセクションバンド幅は 30TB/s である．リンクの冗長性を利用してジョブ単位で任意の 3 次元トーラスネットワークを構成することが可能である．故障ノードを回避することも可能であり, 利便性だけでなく信頼性も高い．並列手法として，超並列性の要請でノード内はスレッド並列，ノード間はプロセス並列というハイブリッド並列が推奨されている．

システムは, システムボードに 4 個のノードが搭載され，筐体に 24 枚のシステムボードが搭載されている．「京」ではこの筐体が 864 台設置され，ピーク性能 10.62PFLOPS，全メモリ量 1.26PB である．OS は Linux をベースとしており，ファイルシステムはグローバルファイルシステムからローカルファイルシステムへステージングをする運用である．評価環境は表 1 の通りである．

表 1　「京」を用いた評価システム

ハードウェア
SPARC64^{TM} VIIIfx, 2GHz, 8 core/CPU, 1CPU/node
浮動小数点演算性能: 128GFLOPS/node
浮動小数点レジスタ: 256 本/core
キャッシュ: L1 - 32KB/core, L2 - 6MB/CPU
メモリ: 16GB/CPU, 0.5B/F
ネットワーク: 3D torus, 5GB/s×双方向 (6 方向)
ソフトウェア
OS: Linux
「京」向け言語開発環境: Fortran，MPI，SSL2(BLAS，LAPACK，ScaLAPACK)

1.3 重点アプリケーション

理化学研究所では，「京」を用いてこれまでの計算機では不可能であったような大規模シミュレーション計算を行い，画期的な研究成果をあげることを目的として，6 本のアプリケーション(以下，重点アプリケーションと呼ぶ.)を選定し,「京」向けの計算性能最適化の作業を行ってきた．この重点アプリケーションは,「京」の汎用性を実証することを目的としているため，分野，計算手法を網羅するように選び出されている．2012 年度 9 月の共用開始時点で，全ての重点アプリケーションについて「京」のフルシステム上で高い実効性能が得られることを確認し，科学的な成果を得るためのプロダクションランが開始できる段階となっている．

これらのアプリケーションの高性能化作業はシステム開発と並行して行われた．また共用開始に先立ち 2011 年 4 月からは試験利用として，広くユーザにもアプリケーションの開発並び成果創出のために，開発中の計算資源の提供を行ってきた．こうしたアプリケーションの高性能化の過程で得られた性能の解析を行うことで，開発中ハードウェア，ライブラリ，コンパイラなどのシステムの評価・検証が行われた．また解析により見いだされた問題点や性能に関する情報は集約し，理研とシステム開発者と共同で議論を行い，アプリケーション開発への知見の蓄積を行うと供に, 必要に応じてシステム開発へフィードバックを行った．

「京」開発プロジェクトの完了に伴う共用開始にあたり, 一連の開発作業で見つかった知見のうち，TLB が実アプリケーション性能へ与える影響に着目し解析した．本解析に用いたアプリケーションは，重点アプリケーションのうち FrontFlow/blue[9]，PHASE[10]，NICAM[11][12][13][14]の 3 種類である．

2. TLB によるアプリケーション性能への影響

本節では，TLB についての説明，並びに TLB がアプリケーションの性能へ与える影響について議論する．

2.1 TLB とは

アプリケーションがメモリアクセスを行う際に，仮想メモリアドレスから物理メモリアドレスへの変換が必要になる．アドレス変換には，主記憶上に存在するアドレス変換テーブルを使用するが, その高速化のために CPU の中にある TLB が使用される．TLB とは，メモリ管理ユニット内のある種のキャッシュであり，アプリケーションが使う仮

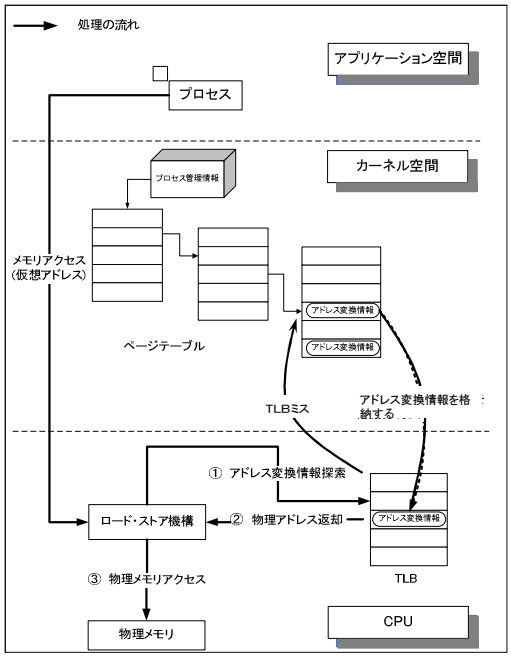

図 1　TLB によるアドレスの変換の流れ.

想メモリ空間と物理メモリ空間の橋渡しをするアドレス変換バッファのことである[1]．図 1 は処理の流れにおける TLB の役割について概念図である．現在，「京」を含めて，多くの仮想記憶をサポートするマイクロプロセッサでは，本機構が使用されている．「京」で用いられている TLB は，sTLB と fTLB の 2 系統があり，sTLB は 256 エントリ，2way セットアソシアティブであり，fTLB は 16 エントリ，フルアソシアティブである．

2.2 TLB ミスによるアプリケーション性能へ与える影響

アプリケーションからメモリへのロード・ストア要求を受けとった CPU は，ロード・ストア先として指定された仮想アドレスに対応する物理メモリアドレスを，TLB に対して求める．TLB 上に仮想アドレスに対応するエントリがあれば，検索結果として対応する物理アドレスが返る．これを「TLB ヒット」と呼ぶ．要求したアドレスが TLB 内にない場合は「TLB ミス」となり，アドレス変換のためにページテーブルを辿っていかなければならない．この手続きには複数個所のメモリの内容を読み取り，そこから物理アドレスを計算し TLB を再構成する時間が必要となる．そのため TLB ミスが頻繁に発生する場合には性能低下の要因となり得る．

「京」では，セットアソシアティブ方式の sTLB とフルアソシアティブ方式の fTLB の 2 種類の TLB が用意されており，ユーザが選択して使用することが可能である．特に，エントリ数が多い sTLB を活用することにより，TLB ミスが軽減し性能向上が期待できる．TLB ミスを回避するために，「京」では，OS の機能としてラージページを調整する機能が用意されている[1]．ラージページとはメモリ管理の基本となる取り扱い単位であり，メモリ管理単位をコントロールすることで，より大きなメモリ空間を網羅することが可能となる．「京」では，ページ単位として，8KB, 4MB, 32MB, 256MB 単位の指定が可能である．またアプリケーション内で使用頻度の高いメモリセグメントと，エントリ数が多い sTLB が使用するラージページサイズを同一のサイズに割り当てることにより，きめ細かく TLB ミスの発生を抑止することが可能である．sTLB は 2way の 256 エントリであり，256 ページを使用するため，ラージページサイズを 256MB に設定した場合は，256MB×256 ページ=64GB のメモリ空間をカバー可能である．「京」では 1 ノードあたりのメモリが 16GB であり，「京」の商用版として開発された FX10 では 32GB, 64GB であるので，全メモリ空間上での TLB ミスの発生を抑えることが可能となり，実効性能の向上が見込まれる．但しメモリアクセスが連続なアプリケーションでは TLB ミスによる性能低下の可能性が少ないため，メモリの有効利用を考慮し，デフォルトページサイズは 4MB としている．

TLB ミスが発生する要因として，先に格納されたアドレス変換情報が，後から格納するアドレス変換情報によって追い出されるというスラッシングがあげられる．アプリケーションの中で，同じエントリに割り当てられるアドレス変換情報が多い区間では，スラッシングが定常的に発生し，性能低下が起こり得る．サイズの異なる配列のアドレス変換情報が区間内にいつか存在するときは，それぞれについて異なるエントリ数が割り当てられるため，実行毎にエントリの相対位置が変化し，アドレス変換情報の追い出しが発生する時としない時が混在する．この時性能は，非定常的に低下する．

図 2 はラージページサイズを全て 256MB に指定した時の実効性能を基準とした，4MB, 32MB で測定した時の性能比，並びに TLB ミス率について，ループ条件を変えて「京」で測定したものである．ループ 1 回転で TLB 1 ラインをアクセスし，配列引数の 2 次元目の増分がライン 1 単位とした時のストライド幅とすることで，メモリアクセスの増分値を変化させ，意図的に TLB ミスを発生させた．これを見ると，TLB ミス率と性能の間には負の相関があることがわかる．図 2 での測定結果をもとに，ラージページサイズ 256MB での実効性能を基準とした性能比と TLB ミス率の関係をプロットした(図 3)．実効性能が 3%低下した時を性能低下の目安値として採用し，その時の TLB ミス率を調べると，4MB の時は 0.0054%，32MB の時は 0.0058%であった．以上から，「京」の場合 TLB ミス率が 0.005%以上の時，性能低下の要因となり得ると見積もられた．目安値を超える TLB ミスが発生する場合，ラージページをより大きく設定するか，アプリケーションの立場からデータ局所性のチューニングについても検討する余地がある．

2.3 性能ばらつき

性能測定を行うと，測定性能が再現しないという現象が見られることがある．これを性能のばらつきと呼び，ばらつきのスケールによって，いくつかに分類可能である．

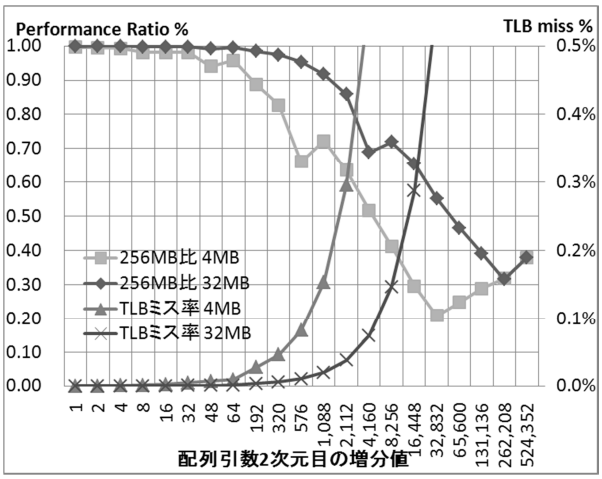

図 2　ループ条件を変えて測定した，ラージページサイズ 256MB を基準とした実効性能比と TLB ミス率．左軸は性能比，右軸は TLB ミス率，横軸はループ条件として，配列引数 2 次元目の増分値を変化させた．

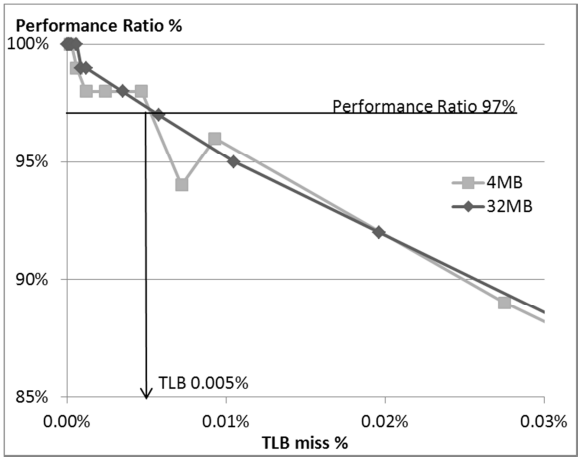

図 3　ラージページサイズ 256MB を基準とした実効性能比と TLB ミス率の関係.
性能比が 3%低下する TLB ミス率は約 0.005%である.

(1) **プロセス毎のばらつき**

並列実行時に性能低下を伴う現象がプロセス毎に発生箇所や頻度が異なる場合，プロセス間のロードインバランスを引き起こす．このロードインバランスは，並列数が増えるほど増大し, 並列性能を低下させる原因にもなり得る.

(2) **ステップ毎のばらつき**

1 つのプロセス内でも性能低下を伴う現象の発生箇所や頻度が変化する場合，長時間走行時の計算性能における不安定性につながる．これは分子動力学や流体解析などの時間発展に伴う繰り返し計算において顕著となり，繰り返し回毎に速いステップと遅いステップが混在するといった現象として測定される.

(3) **実行回毎のばらつき**

長時間計算でのリスタートやアンサンブル平均を取得するために，同等の計算を何度か走行した際に，実行回毎に速い実行回と遅い実行回が混在するといった現象として測定される.

性能ばらつきの原因としては，ロードモジュールへのアクセス集中, データ IO 集中, デーモンなどによるノイズ, キャッシュミスなどが様々な要因が考えられ，アプリケーションのアルゴリズムに起因するものとシステムに起因するものに大別が可能である．キャッシュミスなどのアプリケーションに起因するものについては，該当箇所におけるデータ局所性のチューニングやアクセスパターンのコントロールなどにより解決可能である．システムに起因するものについては，その性能ばらつきを解消すべく，解析結果をシステム開発側にフィードバックを行った．ロードモジュールへのアクセス集中については，ステージング機能により計算ノード直下のローカルディスクにロードモジュールを配置することによって解消した．また OS などのデーモンによるノイズの影響については，影響が大きいと見積もられたデーモンの動作頻度を低く抑え，プロセス間でデーモンの同期を行うことによって，アプリケーションへの影響を最小限に留めることに成功した[15]．いくつかの性能ばらつきの中でも原因究明に時間を必要としたものが，2.2 節で議論した非定常的に発生する TLB ミスによるものであった.

3. TLB ミスによるアプリケーション性能の例

本節では，TLB ミスによるアプリケーションの性能への影響をアプリケーションの実測結果をもとに説明する.

3.1 発生機構

TLB ミス起因の性能低下は，キャッシュミスが引き起こす性能低下に類似しているが，アドレス空間のマッピングで生ずる不整合に起因しているため，アプリケーションから発生原因や発生箇所を特定することは困難なことが多い. また，常に同じ箇所で発生するとは限らないため，時として，性能低下ではなく性能ばらつきとして顕在化することがある．性能ばらつきは TLB ミスだけに起因しないため, 他の要因を抑制した後でないと, 他のばらつきに埋もれて, 原因の特定はおろか，現象の切り出しも困難となる.

2.3 節で論じた性能ばらつきに照らし合わせると，プロセス毎に TLB ミスの発生箇所や頻度が異なる場合に(1)のロードインバランスが測定される．アプリケーションの区間内で確保されるエントリ数が均一でないことに起因し，スラッシングが非定常的に発生する時など，発生回数が比較的少なく発生箇所やその頻度が変化する場合(2)の長時間実行時の性能ばらつきが観測され，ライブラリの動的リンクなどの何らかの割り込みなどにより TLB が汚されると(3)の実行毎の性能ばらつきが観測される．この発生箇所や頻度の不確定性は，アプリケーション由来のロードインバランスなどに起因することもあるが，時として OS/ライブラリなどのシステム由来のこともある.

2 節で議論したように,「京」では，TLB ミスをなくすために，ページサイズを最適化することで，完全に抑制することが可能である．具体的には，ラージページサイズを 256MB に設定することで, 全てのメモリ空間を網羅可能となり，原則として TLB ミスは発生しなくなる.

ラージページサイズを大きくするとアプリケーションが使用するメモリ量も増大する．全ての TLB の設定を 256MB にすることで, 全スレッド数分のページサイズも増えるため, 最大で 2GB 程度のメモリ使用量が増える計算になる．このためアプリケーションによっては，動作に必要となるメモリ量の制限からページサイズを細かく設定する必要がある．しかしアプリケーション毎に頻繁に使われるメモリセグメントのみラージページサイズを大きく設定することで，アプリ全体のメモリ使用量を低く抑えることが可能である.

3.2 FrontFlow/blue におけるプロセス間の性能のばらつき

FrontFlow/blue は非圧縮性流体の非定常流動を高精度に予測可能な Large Eddy Simulation(LES)にもとづいた汎用流体解析プログラムである[9]．形状適合性に優れた有限要素法による離散化を採用し，ファンやポンプ等の流体機械や複雑形状周りの非定常乱流解析および流れから発生する騒音の予測が可能である．

図 4 は，FrontFlow/blue を 512 プロセスを用いて，同じ計算を 3 回実行した際の，運動方程式を計算する区間 vel3d1 の実行時間をプロセス毎に並べたものである．測定で用いた系は，領域分割に際して，各プロセスが受け持つ有限要素数を誤差数%程度とおおむね均一に抑えているため, 演算時間や通信時間のばらつきは少ないと想定される．左図を見るとプロセス毎の実行時間のロードインバランスは最大で 4 倍近いことがあり，その分布も実行回毎に異なることがわかった．プロセス毎のロードインバランスは，集団通信やバリア同期により．待ち時間として蓄積するため，この区間の性能は 12.0[s] → 46.0[s]と 3.83 倍の性能低下として測定されることになる．

ハードウェアカウンタ情報を精査すると TLB ミス率が高かったため，ラージページサイズを変更して再度測定を行った．本区間で用いられる配列はヒープセグメントに置かれる配列であったため，使用メモリ量を極力増やさないように，ヒープセグメントと sTLB への割り当てサイズのみを 32MB から 256MB へと変更したところ，右図に示すように，プロセス毎の実行時間及び実行毎の実行時間は一致するようになり，全体性能も向上した．

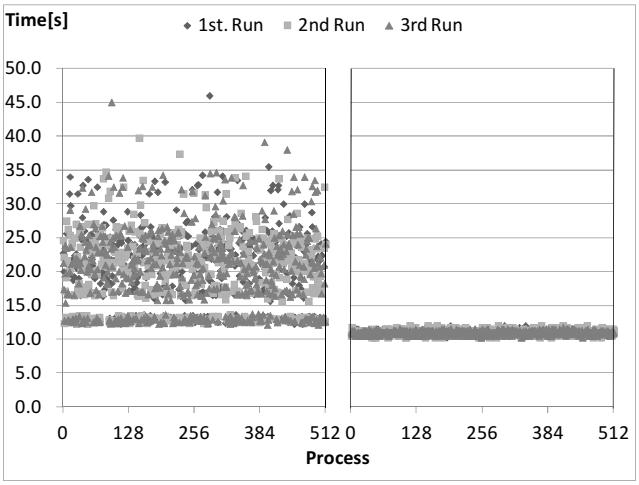

図 4　FrontFlow/blue の区間 vel3d1 における実行時間分布．同じ計算を 3 回実行し, プロセス毎, 実行回毎に測定した．左図はラージページサイズを全て 32MB に設定し，右図はヒープセグメントと sTLB を 256MB に設定して実行した．

3.3 PHASE での性能のばらつき

PHASE は, 擬ポテンシャルと密度汎関数法によるナノ材料第一原理分子動力学プログラムであり，局在基底ではなく平面波基底を用いることにより，分子から固体まで多くの物質に対して高精度な電子状態計算が可能である[10]．この計算には，多様な行列計算，高速フーリエ変換，固有値解析などが含まれ，また多軸並列という複雑な並列手法を用いているため，多種の集団通信が用いられている[16]．

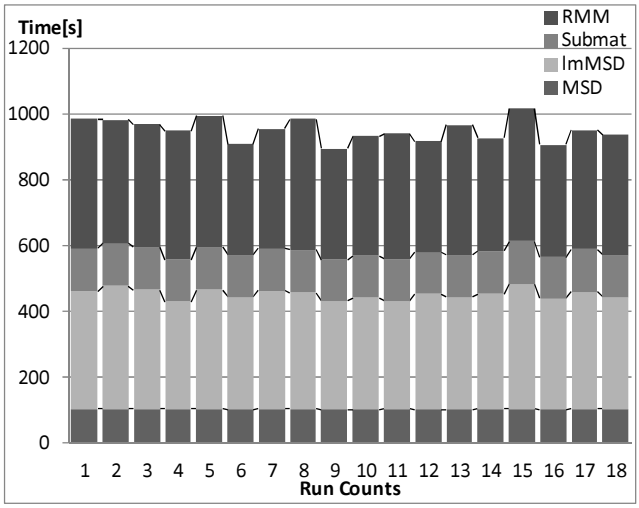

図 5　PHASE における実行回毎の実行時間．4 ソルバの実行時間積み上げグラフである．SiC 12,544 原子 12,288 プロセスで計算し，ラージページサイズは全て 4MB に設定した．

図 5 は，PHASE を用いて，SiC12,544 原子を 12,288 プロセスで電子状態計算したときの実行時間を 4 種類のソルバについて積み上げたグラフである．ラージページサイズを 4MB に設定し，同じ計算を 18 回実行してその内訳を比較した. 実行時間ばらつきは, 初回計算される MSD と Submat がそれぞれ，2.34%，2.58%であるが，2 回目に計算される lmMSD は 14.84%，最後に計算される RMM は 18.13%と増大した．

図 6 は，図 5 の lmMSD と Submat の 2 種類のソルバを繰り返し単位として選び, 10 回の長時間収束計算を行った際の実行時間の変化である．DGEMM による行列-行列積，FFT，固有値計算など計算特性毎に 11 区間に分割し積み上げている．左グラフはラージページサイズを全て 4MB で実行したものである．繰り返し回毎に，実行時間が増加する区間があることが分かる．左図の 5step で時間が増えた箇所は, DGEMM を用いて行列-行列積を計算する 2 区間であり, TLB ミス率はそれぞれ 0% → 0.0001%, 0.0002%とわずかではあるが増加していた. 8step と 10step で実行時間が増加した区間は固有値計算を行う区間であり，TLB ミス率は 0.0008% → 0.001%とやはり悪化している. PHASE では，問題規模 N に対して $O(N^3)$でメモリサイズが増大する．このため，ラージページサイズを大きくすることで，メモリが不足することがある．このため右図では，ヒープセグメントとデータセグメントのみ 256MB に設定し，sTLB の使

用するページを 256MB に割り当てた．メモリ使用量節約のためにスタックセグメントとスレッドセグメントは 4MB に据え置いた．ラージページサイズを最適化することで，実行時間のばらつきが 13.83% → 1.31%に減少し，性能のばらつきを消すことができただけでなく，全体性能も平均値で比較して 5.78%程度向上している．計測区間がループ単位でなくサブルーチン単位と大きいため，TLB ミス率は希釈され，目安値 0.005%以上として測定されていないが，ラージページの設定を変えることで，性能ばらつきが解消するだけでなく全体性能も向上しているこがわかる．

PHASE における性能のばらつきの特徴として，

(1) **該当箇所にアルゴリズム上の問題が見つかることは少ない．**
(2) **走行回毎にある区間が遅くなったとき，同時に別の区間が速くなることがある．**
(3) **部分的にコード修正を行うと，全く異なる別の区間が遅くなることがある．**

などの現象が見られた．これらは TLB ミスによって説明が可能である．(1)については，TLB ミスはアドレス変換で起こるキャッシュミスであるため，ループ構造やデータの並びを変えるなどの，データアクセスパターンを変更することでは，性能改善が望めないことがあり，問題の特定の難しさがある．(2)については，走行回毎にアドレス配置が変わることで発生する TLB ミスが，増減することなく，アプリケーション内で移動したと解釈可能である．(3)については，コード改変によって，アドレス配置が変化し，別の箇所での TLB ミスが誘発されたと解釈可能であり，実際，PHASE において固有値計算の箇所を ScaLAPACK から別の高速ライブラリに置き換えた所，全く別の区間である行列-行列積を行う DGEMM 計算の性能が大幅に低下するなどの現象として確認された．

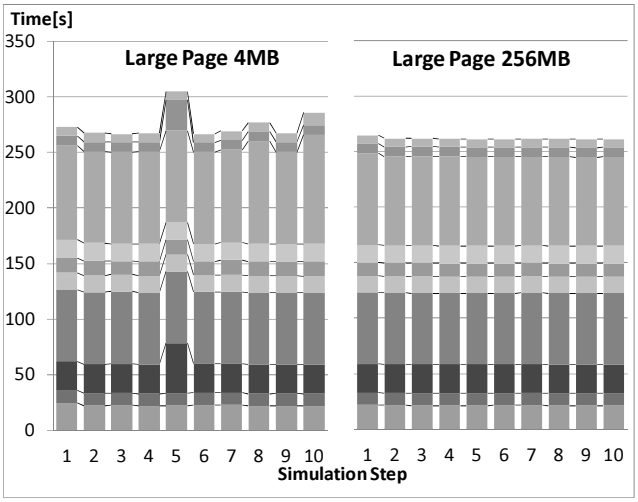

図 6　PHASE における長時間実行での実行時間．計算特性 11 区間に分類した積み上げグラフである．SiC 12,544 原子を 12,288 プロセスにて実行し，lmMSD と SUBMAT ソルバを 10 回繰り返した．左はラージページサイズを全て 4MB に，右はヒープセグメント，データセグメント，sTLB を 256MB に設定して実行した．

4. マスク付 SIMD 機能による TLB ミス

本節では，SIMD 拡張された条件付きストアを利用するコンパイラの最適化機能(以下，マスク付 SIMD 機能と呼ぶ．)によって発生し得る TLB ミスがアプリケーション性能へ与える影響について議論する．

4.1 NICAM での性能事例

NICAM（Nohydrostatic ICosahedral Atomospheric Model）は，正二十面体非静力学大気モデルを採用した，全球を対象とする大気大循環モデルの一つである[11][12][13][14]．地球シミュレータ(ES)上で開発されてきた経緯により，ES などのベクトルプロセッサにおいて高性能が得られるようにコーディングされている．一方で，様々なアーキテクチャでの大規模実行の動作実績があり，ポータビリティの高いアプリケーションである．

本アプリケーションの雲微物理過程の計算を行う区間である NSW6 のアプリケーション性能を測定した結果が，図 7 である．差分法計算ではあるが，物理過程の計算を行うために要求 B/F が比較的小さく，最適化を促進することで演算効率が向上する可能性が高い．しかしループボディが大きいため SIMD などによる最適化の適用が難しい特徴がある．この区間をシステムモードとユーザーモードを分類し，命令コミット種類を調べると，オリジナル実行時の計算時間は 4.18[s]に対して，コンパイラによる最適化機能であるマスク付 SIMD 機能を有効にすることで，227.27[s]に増大した．経過時間の大部分はシステムの処理に要した時間であり，中でも整数ロードキャッシュアクセス待ち，命令フェッチ待ちといったアドレス計算に起因する時間が 60%程度占めていることがわかる．またこのときの TLB ミ

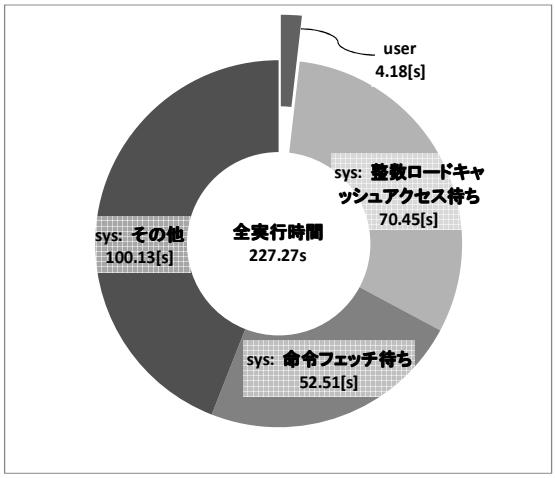

図 7　NICAM の区間 NSW6 における経過時間内訳．

ス率は 1.37%であり，性能低下の目安となる 0.005%を大幅に上回っていた．更に，この TLB ミス率は，ラージページサイズの調整により低減出来なかった．

4.2 NICAM における TLB ミスの発生原理

アプリケーション開発はシステム開発と並行して行われるため, 導入された新機能についての評価が必要になる．NICAM における性能低下は，コンパイラ上で新規に開発中の機能に起因すると解析された．本機能は，IF 分岐で囲まれたデータのストアに対して，マスク付 SIMD 機能を適用するものである. この機能は, ベクトルマシンにおいて，ループのベクトル化率を向上させるため，古くから用いられてきた手法である．IF 文の真偽率によって，その効果が変わることが知られるが，命令スケジューリングが促進されるため，多くの場合，ループ性能の向上が見込まれる[6][17]．実アプリケーションで評価を実施することで，以下の条件で，性能低下を生じ得ることが分かった．

(1) **最内ループ内で不変となる条件文を条件式とする IF 文が存在する.**
(2) **IF 文内に ALLOCATABLE 配列により領域を確保する.**
(3) **ALLOCATABLE 配列が IF 文内で定義される.**
(4) **ループボディが大きい.**
(5) **ループボディに複数の IF 文が存在する.**

サンプルコードを図 8 に示す．原因は，物理メモリ上に allocate されていない配列について，SIMD 拡張された条件付き命令により指定された仮想ストア先が見つからず，TLB ミスに伴うカーネルエミュレーションが発生したためと分析された(図 9)．ループボディが小さい時は，ループ if 展開による最適化が行われ，本現象は発生しない．しかしループボディが大きい時，if の真偽毎，つまり最大で $2^{if\ の数}$回のループが生成する．これを回避するため if 展開は行われず本現象が発生する．本現象は，実在しないアドレスに対してストアを試みることにより生ずる TLB ミスであるため，ループ内で毎回必ず発生し，ラージページサイズをコントロールすることでの除去は不可能である．またアプリケーションを異常終了させないために OS エミュレーションが実行されるケースがあり高コストになる上，実行時に初めて顕在化するため，コンパイル時に除去可能か判断することは難しい．

```
!flg = false
if (flg) then
   allocate (a(100))
endif

do i=1,n
   ・・・
   if (flg) then
      a(i) = 2
   endif
   ・・・
enddo
```

図 8　マスク付 SIMD 機能により，
TLB ミスが発生するコード例.

本現象は，アプリケーション側がコーディングによって対応するか, システムで用意されている機能を使うことで，回避することが可能である．問題は，マスク処理による allocate されていない配列へのデータストアであるため, アプリケーション側の対策として，allocate を IF 文の中で使用しないという回避策が考えられる．しかし，メモリ使用量の問題で必要最小限に配列を確保するという要請からこの方法を採用している場合は，システムで提供されている機能を使う方法が推奨される. 本機能は, コンパイル時に，該当箇所が性能低下を引き起こす可能性があることを，情報として出力する機能である．この情報をもとに，該当箇所，もしくは数が多い場合は性能低下が測定される箇所を更に絞り込み，その箇所のみ allocate を行う IF 文を削除したり，提供された指示行などにより，新機能のマスク付 SIMD 機能を局所的に無効化するなどの対策が可能である.

4.3 PHASE を用いたマスク付 SIMD 機能の評価

3.3 節で議論したアプリケーション PHASE は, 行列計算，FFT，対角化など多様な処理を行う．大規模な系の計算では，メモリ削減を目的として，処理区間毎に計算条件に合わせた配列をその都度確保する方法が用いられている．このため, 本現象が発現しやすいアプリケーションと言える.

実際，該当箇所を自動で調べるコンパイラの機能を利用して, 調査すると 143 ファイル 20 万行ほどのコードの中に，163 箇所の該当箇所が見つかった．性能測定を行った所，非局所ポテンシャルの行列計算に関する比較的規模の大きいループ 1 区間においてのみ，実行時間が 2.94[s]増加した箇所が見つかり，TLB ミス率が 0.0001% → 0.1163%と上

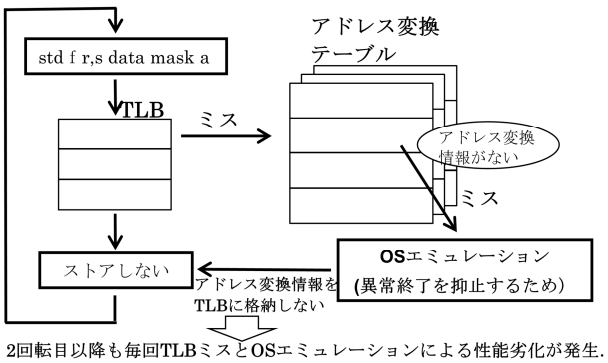

図 9　マスク付 SIMD 機能によるデータストアの概念図.
物理メモリ上に割当てられず MASK
が off の時に TLB ミスが発生する.

昇していた．しかし，その他の大部分の区間については，性能と TLB ミス率の明確な因果関係はみられなかった. 4.2 節の 5 条件を全て満たすループが少なかったものと推察される．全体でも TLB ミス率は 0.0004% → 0.0008%と増加しており，実行時間が 625.0[s]の所 3.7[s]増えていたが，その増加の割合は 0.59%程度と，ストアに対するマスク付 SIMD 機能を使わない時とほぼ同等の性能が出ているといえる．

5. 考察

今回「京」での測定を用いて TLB ミスのアプリケーションの性能へ与える影響について議論した．この現象は，今後，多くの仮想記憶をサポートするマイクロプロセッサにおいて, メモリサイズが増大し, 並列数が多くなるに従い，より顕在化するであろう問題といえる．むしろ既に顕在化しているが，その原因特定まで至っていないケースも多いと思われる．

ハイパフォーマンスコンピューティングでは，実行効率だけでなく，設置面積，電力効率も含めて設計する必要がある．プロセッサの設計段階でダイをシュリンクさせる必要があるのもこのためである. ダイをシュリンクする場合，交換条件としてアーキテクチャやインターフェースに制限を設けられ，機能や性能に制約が発生する場合がある．今回の TLB ミスも, このような制約が要因となって発生した. 「京」では，これらの制約に対して，ハードウェアの性能を極限まで引き出すために，ソフトウェア側で補足できる手段が用意された.

「京」では OS の機能であるラージページサイズをきめ細やかにコントロールすることで，性能への影響を完全に制御することが可能となり，新規に見つかった現象についても，アプリケーション，システム側での対処方法が見いだされた．これらは，ハードとシステムソフト，そしてアプリケーション開発者が協力して成し遂げた事に意義がある．しかしこのラージページによる解決法も，ユーザが使用可能なメモリ領域が圧迫される懸念があり，一長一短である．今後，エクサスケールのスーパーコンピュータ開発に向けて，システムソフトやアプリケーションが共同で議論してきた知見をハードウェア開発の現場にフィードバックさせることで，より最適な設計・開発に結びついていくと期待される．

6. まとめ

アプリケーション性能は，様々な原因で変化する．今回その中でも TLB ミスによる性能低下の可能性, そして回避策について議論した．TLB ミスは，スラッシングなどで常に起こるものとそうでないものに分類でき，後者は，性能低下として測定されるだけでなく，ロードインバランス，シミュレーションステップ毎，実行回毎の性能ばらつきとして測定されることが多い. また TLB ミスはアドレス情報の変換で生ずるため，データの連続性を作り出す以外の方法で，アプリケーションの立場でその原因を特定，解決するのは難しいといえる．

今回，FrontFlow/blue, PHASE で見られた性能のばらつきは，全てラージページサイズをコントロールすることで，抑制することに成功した．また新規開発中の機能の評価段階にて NICAM で見られた性能低下については，アプリケーションの立場で取るべき対策が知見として見いだされた. 更に，システム側にフィードバックすることで，アプリケーション開発をサポートすべく，性能低下が起こり得る箇所を自動で調べるというコンパイラの機能が用意された. 4.3 節の PHASE での実測例を見ると，実アプリケーションでは，NICAM ほど極端に性能低下が見られることは少ないと想定されるが，性能低下の可能性についての知見が得られたことは意義がある．

なお TLB ミスに起因する性能のばらつきは, 並列数に比例して発生頻度が高まるため，並列数が大きいほどその影響が大きくなると想定される．PHASE は，今回の解析により，TLB ミスが性能へ与える影響が大きいアプリケーションであると判断されたため，「京」の一部である 73,728 プロセス 9.44PF のシステム規模で, ラージページサイズを調整して測定し直した．その結果 SiC の螺旋転位という 20,440 原子系の実問題に対して，実行効率が，14.53%から 21.67%，実効性能としては，1.37PF から 2.05PF へと約 1.5 倍性能が向上した.「京」フルシステムでは実行効率 20.18%, 実効性能 2.14PF を達成することができ, アプリケーションの主要部に FFT による全体通信を多く含むものとして，画期的な性能を得ることができた.

謝辞 本報告に際し，システムソフトウェア開発者の立場で御討論頂いた，青木正樹氏，杉山浩一氏を始めとする富士通株式会社次世代 TC 開発本部の皆様，並びに理化学研究所計算科学研究機構運用技術部門の諸氏，理化学研究所計算科学研究機構に常駐して「京」の運用支援に携われている井上晃氏，坂本清隆氏，若林大輔氏を始めとする富士通株式会社 SE の皆様に感謝します．またスーパーコンピュータ「京」の性能向上のために，アプリケーションを利用させて頂いた，FrontFlow/blue, PHASE, NICAM の開発者に感謝します．本論文の結果は，理化学研究所計算科学研究機構が保有するスーパーコンピュータ「京」の試験利用によるものです.

参考文献

1) Silberschatz, A., Galvin, P. B., Gagne, G.: Operating System Concepts, 8th Edition, ISBN 0-470-12872-0, John Wiley & Sons Inc.

(2008).
2) Maruyama, T.: SPARC64 VIIIfx: Fujitsu's New Generation Octo-core Processor for Peta Scale Computing., Hot Chips 21 (2009).
3) Maruyama, T.: SPARC64 VIIIFX: A New-Generation Octocore Processor for Petascale Computing, IEEE micro, Vol.30, No.2, pp30-40 (2010).
4) SPARC International, The SPARC Architecture Manual (Version 9), Prentice-Hall (1994).
5) Sparc Joint Programming Specification (JPS1): Commonality, architecture manual., Sun Microsystems and Fujitsu Ltd. (2002).
6) SPARC64VIIIfx Extensions, Fujitsu Ltd., architecture manual (2008).
7) Ajima, Y., Sumimoto, S. and Shimizu, T.: Tofu: A 6D Mesh/Torus Interconnect for Exascale Computers., IEEE Computer, pp36-40 (2009).
8) Toyoshima, T., ICC: An interconnect controller for the Tofu interconnect architecture., Hot Chips 22 (2010).
9) FrontFlow/blue, http://www.ciss.iis.u-tokyo.ac.jp/rss21/theme/multi/fluid/fluid_softwareinfo.html
10) PHASE, http://www.ciss.iis.u-tokyo.ac.jp/riss/project/device/
11) Satoh, M., Matsuno,T., Tomita, H., Miura, H., Nasuno, T. and Iga, S.: Nonhydrostatic Icosahedral Atmospheric Model (NICAM) for global cloud resolving simulations, Journal of Computational Physics, the special issue on Predicting Weather, Climate and Extreme events, 227, pp.3486-3514, doi:10.1016/j.jcp.2007.02.006 (2008).
12) Tomita, H., Miura, H., Iga, S., Nasuno, T. and Satoh, M.: A global cloudresolving simulation: Preliminary results from an aqua planet experiment, Geophys. Res. Lett., Vol.32, L08805, doi:10.1029/2005GL022459 (2005).
13) Satoh, M., Tomita, H., Miura, H., Nasuno, T. and Iga, S.: Develpment of a global cloud resolving model - a multi-scale structure of tropocal convections., J. of Earth. Sim., Vol.3, pp.11-19 (2005).
14) Tomita, H. and Satoh, S.: A new dynamical framework of nonhydrostatic global model using the icosahedral grid, Fluid Dyn. Res., Vol.34, pp.357-400 (2004).
15) 師尾 潤, 山田 雅彦, 加藤 丈治: スーパーコンピュータ「京」のオペレーティングシステム, 雑誌 FUJITSU, Vol.63, No.3, pp273-279 (2012).
16) 黒田 明義, 長谷川 幸弘, 寺井 優晃, 井上 俊介, 市川 真一, 小松 秀実, 大井 憲行, 安藤 琢也, 山崎 隆浩, 大野 隆央, 南 一生: ナノ材料第一原理分子動力学プログラム PHASE の京速コンピュータ「京」上の計算性能最適化, ハイパフォーマンスコンピューティングと計算科学シンポジウム論文集, Vol.2012, pp.144-152 (2012).
17) Aho, A. V., Lam, M. S., Sethi, R., Ullman, J. D.: コンパイラ[第２版] ～原理・技法・ツール～, Information & Computing, 別巻 38, ISBN978-4-7819-1229-5, サイエンス社 (2009).