Vol.2010-HPC-124 No.4
2010/2/22

メモリバンド幅に着目したマルチコアノード上のアプリケーション最適化

三木康次[†1] 原田知徳[†1] 朴泰祐[†1,†2]
塙敏博[†1,†2] 三浦信一[†2]

プロセッサのマルチコア化は，チップ当たりのピーク性能の大幅な向上をもたらしているが，その一方でプロセッサチップ当たりのメモリバンド幅の向上はコア全体の要求には応えられていない．この状況により，特にメモリアクセス率の高い HPC アプリケーションにおけるメモリウォール問題は今後ますます深刻化すると考えられる．本研究では，実アプリケーションにおけるメモリ性能要求の観点から，現在のマルチコアプロセッサがどの程度その性能を活かせているかを検証する．それに基づき，アプリケーションに応じて適切な利用コア数が存在することを指摘し，性能最適化への指標を与える．高メモリバンド幅を要求するアプリケーション例として Lattice QCD について，超並列クラスタ T2K-Tsukuba を用いて検証した結果，ノード上の 16 core のうち 8 core までの利用で性能向上は頭打ちになり，システム管理デーモンとの関係で，むしろそれ以上の利用では性能が低下することが示された．

Performance Optimization based on Memory Bandwidth Characteristics on Multi-core Node

Koji Miki,[†1] Tomonori Harada,[†1] Taisuke Boku,[†1]
Toshihiro Hanawa[†1] and Shin'ichi Miura[†1]

Recent trend of core count increasing on multi-core processor achieves very high peak performance on a chip. On the other hand, the memory bandwidth improvement per chip has been not sufficient to support such a performance enhancement on a chip. This fact causes more serious memory wall problem especially for scientific code which requires high memory bandwidth. In this research, we evaluate the memory bandwidth bottleneck on real applications to provide some guideline to optimize the utilization of cores on multi-core node on PC cluster. Examining Lattice QCD code which requires high memory bandwidth, it is shown that the optimal core count on 16-core computation node on large scale PC cluster T2K-Tsukuba is 8 but 16. Moreover, according to the additional load by OS, it is observed that more number of cores depress the performance in total.

1. はじめに

現在，HPC (High Performance Computing) 分野ではクラスタ型の高性能計算機が多用されている．しかし，昨今のプロセッサの急速なマルチコア化により演算性能が飛躍的に向上したため，メモリアクセス性能が並列処理のボトルネックとなる場面が目立ってきた．特に，1 台の計算ノード上に複数のソケットを搭載する場合，単一のメモリシステムではそのバンド幅要求を満足できず，従来の AMD Opteron に加え，Intel Nehalem 以降の IA32 アーキテクチャでも，NUMA (Non-Uniform Memory Access) アーキテクチャが採用され，マルチソケット構成でもソケット当たりのメモリバンド幅を確保している．これらの NUMA 型のプロセッサ配置により，各プロセッサからローカルメモリへの独立したメモリアクセスが可能となり，ノード全体でのメモリアクセス性能が向上した．しかし，各プロセッサでは，複数の CPU コアが同一のメモリを共有しており，同一プロセッサ内でのメモリアクセスではメモリバンド幅が性能ボトルネックとなってしまう可能性がある．特にメモリバンド幅要求の高いアプリケーションにおいては，演算処理に必要以上の CPU コアを用いても，期待する性能向上が見込めないと予想される．今後，コア数の増加傾向はさらに続き，メモリバンド幅の向上率を上回ると予想されるため，演算性能とメモリ性能の乖離はさらに大きくなると考えられる．したがって，マルチコア環境における並列処理では，演算並列化のための CPU コア数を適切に設定し，それにより演算に使用しない余剰 CPU コアが生じた場合，それらを演算以外の処理に使用し，プロセッサに搭載された CPU コアを有効に活用することによって，アプリケーション全体での性能を向上させることが有効であると考えられる．

本研究の目的は，実際に HPC 分野で使用される科学計算アプリケーションについて，実効メモリバンド幅とアプリケーション性能の関係を明らかにし，システムに応じた並列性能が最適になる CPU コア数を解明することである．また，このような解析に基づき，並列通信や OS 等のシステムソフトウェアの総合的最適化に余剰コアを投入し，システム全体の

†1 筑波大学大学院 システム情報工学研究科
Graduate School of Systems and Information Engineering, University of Tsukuba
†2 筑波大学 計算科学研究センター
Center for Computational Sciences, University of Tsukuba

性能を改善する指標を得ることも目的である．

2. マルチコア／マルチソケット環境におけるメモリウォール問題

本章では，マルチコア／マルチソケット環境における問題点として，演算性能とメモリアクセス性能の乖離によって起こるメモリウォール問題について述べる．

2.1 メモリウォール問題

マルチコア環境では，複数の CPU コアでの並列処理により高い性能を実現することができるが，並列処理を行っても効果的な性能向上が得られない場合がある．その原因として，メモリウォール問題があげられる．メモリウォール問題とは，演算性能に比べメモリのデータ供給能力が不十分であり演算ユニットへのデータ供給が間に合わなく，処理性能が向上しない問題である．特に近年，プロセッサの演算性能の飛躍的な向上により，演算性能とメモリアクセス性能の差が拡大し，メモリウォール問題はさらに深刻になっている．特に，キャッシュヒット率の低いアプリケーションやアルゴリズムにおいては，この問題は顕在化する．また，マルチソケット構成においてノード全体のメモリバンド幅を確保するために AMD,Intel 双方の IA32 プロセッサにおいて NUMA 型アーキテクチャが利用されるようになってきた．しかしながら，HPC 分野で扱われるような大規模科学技術計算では，使用するデータセットのサイズが各ソケットのホームメモリサイズより大きいことがあり，実行プロセスはソケット（ホーム）外のメモリへアクセスしなければならなくなる局面では，メモリアクセス性能が性能ボトルネックとなってしまう可能性が高い．そのため，このようなメモリバンド幅がボトルネックとなるような状況では，並列化のために必要以上に CPU コアを投入しても，十分な性能向上は望めないと考えられる．

よって，効果的なプログラムの並列化を行うためには，対象プログラムの要求するメモリアクセス性能から演算並列化のための必要十分な CPU コア数を設定し，メモリアクセス性能と演算性能のバランスのとれた並列化を行う必要がある．必要以上のコアの投入は，ソケット内でのメモリトランザクションや共有キャッシュアクセス制御のオーバヘッドを増加し，場合によっては NUMA アーキテクチャ上のキャッシュコヒーレントトランザクションの増加を引き起こす可能性もある．物理的な CPU コア数が，最適な性能を与える CPU コア数を上回る場合，使用すべきでないコアが存在することになる．本研究ではこのようなコアを「余剰コア」と呼ぶ．今後，プロセッサに搭載される CPU コア数がさらに増加し続けた場合，アプリケーションによっては余剰 CPU コアが生じる可能性はさらに高まると予想される．しかし，このような余剰 CPU コアを通信の補助や OS のデーモンのような補助プロセス，さらに性能最適化のためのプロファイリング等，本来の演算外の処理に投入し有効に利用することにより，総合的な性能向上を図れる可能性がある．

したがって，アプリケーションとシステムに応じた最適 CPU コア数を見つけ出し，余剰 CPU コアを通信及び補助的な処理に投入することを，ユーザの手を煩わせずに行うことができれば，マルチコア環境における並列処理性能の向上が可能であると考えられる．

3. 評 価 実 験

本章では評価実験の概要，評価に用いるシステム及びベンチマークについて述べる．

3.1 実験対象システム：T2K-Tsukuba

本研究の評価実験には，T2K-Tsukuba[1),2)] を用いた．T2K-Tsukuba は筑波大学計算科学研究センター[3)] において現在稼働中である大規模 PC クラスタである．構成を表 1 に示す．各ノードに 4 つの CPU コアを持つ AMD Quad-Core Opteron(2.3GHz) を 4 ソケット搭載している．また，各ソケットは Hyper Transport リンクにより結合されており，ノード内は NUMA アーキテクチャとなっている．

表 1 T2K-Tsukuba のシステム構成

構成	4 ソケット（16CPU コア）
CPU	AMD Quad-Core Opteron8356 2.3GHz （Barcelona）x 4
L1 cache	64KBx4（命令）
	64KBx4（データ）
	（各 CPU コア独立）
L2 cache	512KBx4（各 CPU コア独立）
L3 cache	2MB（CPU コア間共有）
総メモリ容量	32GB DDR2 667MHz （ソケット当たり 8GB）
理論総メモリバンド幅	42.7GB/s
理論ピーク性能	147.2Gflops/node

3.2 評価対象ベンチマーク

本節では，評価実験に用いるベンチマークについて述べる．

3.2.1 STREAM Benchmark

STREAM ベンチマーク[4)] は，表 2 に示すような単純なベクトル処理演算により，メモリのロード／ストア性能を測定するベンチマークである．本ベンチマークはプロセッサのキャッシュ効果を反映させないように，キャッシュサイズを超えるサイズの配列を用いる．

本実験では，T2K-Tsukuba の 1 ノード (16CPU コア) について STREAM ベンチマー

表 2 STREAM の演算

COPY	$a(i) = b(i)$
SCALE	$a(i) = q * b(i)$
ADD	$a(i) = b(i) + c(i)$
TRIAD	$a(i) = b(i) + q * c(i)$

クによりメモリアクセス性能の評価を行った．実効メモリバンド幅は，4,8,16CPU コア利用時について測定した．CPU コアとメモリの割り当ては表 3 に示した numactl オプションにより，複数のプロセスが各ソケットに均等に配置され，かつ各プロセスが参照するメモリは NUMA 上でそのプロセスの実行ソケットをホームとするように配置した．複数プロセスの立ち上げには MPI を用いたが，STREAM ベンチマークを実行する各プロセス間に通信は発生せず，Embarassingly Parallel なプロセスとしてベンチマークの最初と最後だけで同期を取る．実効メモリバンド幅の測定には PAPI を用いた．表 3 に STREAM ベンチマークの実行環境を示す．

表 3 メモリアクセス性能評価の実行環境

ベンチマーク	STREAM MPI version
使用ノード数	1 ノード
MPI プロセス数	4, 8, 16
Elements	8000000
使用総メモリ量	3.6GBytes
numactl オプション	SLOT=$($MYRANK % $NSLOTS)
	numactl –cpunodebind=$SLOT –localalloc
	(MYRANK:プロセス番号, NSLOTS:ノード内の CPU ソケット数=4)

3.2.2 Lattice QCD

QCD(Quantum ChromoDynamics:量子色力学) とは素粒子物理学分野の基礎理論であり，素粒子間の強い相互作用について記述されている．Lattice QCD(格子量子色力学) は，QCD での 4 次元時空におけるクォーク場とグルオン場を計算機により解析可能にするために 4 次元離散格子に置き換え，格子点に変数を定義した QCD である．Lattice QCD はその演算当たりのメモリアクセス量が多く，また演算回数そのものも膨大であることが知られている．本実験では Lattice QCD 中で頻繁に用いられる D-1 を求めるベンチマークプログラムである solver_bench を使用する．solver_bench は連立方程式を解くプログラムであり，並列化の際には 4 次元離散格子を部分格子に分割し，それぞれのブロックを各演算ユニットに割り当てる．このとき，それぞれのブロック間では通信によるデータの受け渡しが必要である．

本実験では，T2K-Tsukuba の 32 ノードを用いて，ノードあたり 4,8,16MPI プロセスの場合について，solver_bench の実効メモリバンド幅を PAPI により測定した．全体の問題サイズは一定であり，これが並列プロセス数（ノード当たり 4, 8, 16 プロセスの場合，それぞれ全体で 128, 256, 512 プロセス）に均等に分配される．また，実行に際してメモリアクセス性能評価と同様に numactl オプションにより，効率のよいメモリアクセスができるようにノード内で並列プロセスを 4 つのソケットに均等分配し，各プロセスのメモリはそのソケットをホームとするよう配置した．さらに，並列処理を実行する全てのノードで定期的にバックグラウンドプロセスを同時実行して測定を行い，性能への影響を調査した．バックグラウンドプロセスには，T2K-Tsukuba の各ノードで標準動作している，ユーザプロセスが使用しているメモリ量を調べる memchk と呼ばれる daemon プロセスを使用した．memchk は定期的にメモリ使用量を監視している daemon であり，T2K-Tsukuba でのジョブ実行時に動作の有無を指定することができる．表 4 に QCD ベンチマークの実行環境を示す．

表 4 QCD ベンチマーク性能評価の実行環境

アプリケーション	solver_bench
使用ノード数	32 ノード
MPI プロセス数	128,256,512
ノード当たりの使用 CPU コア数	4,8,16
QCD の問題サイズ (x,y,z,t)	(32,32,32,64)
ブロック当たりの格子サイズ	4
daemon	memchk
numactl オプション	SLOT=$($MYRANK % $NSLOTS)
	numactl –cpunodebind=$SLOT –localalloc
	(MYRANK:プロセス番号, NSLOTS:ノード内の CPU ソケット数=4)

3.3 PAPI

Performance Application Programming Interface (PAPI)[5] はハードウェアパフォーマンスカウンタにアクセスするための API である．現在のプロセッサの多くにはハードウェアパフォーマンスカウンタと呼ばれるプロセッサでの処理イベントのカウントを行う専用のレジスタが搭載されている．これらのカウンタへのアクセスを容易にするためにカーネルに perfctr と呼ばれるパッチを当て，パフォーマンスカウンタへのアクセス方法を統一的に提供するシステムが PAPI である．PAPI は様々なプロセッサファミリーに委嘱されており，

Vol.2010-HPC-124 No.4
2010/2/22

浮動小数点演算命令数，メモリアクセス回数等，性能チューニングの基礎となる重要なパフォーマンスデータをユーザレベルで容易に取得することができる．本研究では，これらのカウンタから表 5 のようなイベントを測定し評価を行う．

表 5 PAPI により取得するイベントカウンタ

HYPERTRANSPORT_LINK[0:3]:DATA_DWORD_SENT	HyperTransport[0:3] で送信されたデータ数
DRAM_ACCESSES_PAGE	Dram Access 関係のカウントの取得
:HIT	DCT0 のページヒット
:MISS	DCT0 のページミス
:CONFLICT	DCT0 のページコンフリクト
:DCT1_PAGE_HIT	DCT1 のページヒット
:DCT1_PAGE_MISS	DCT1 のページミス
:DCT1_PAGE_CONFLICT	DCT1 のページコンフリクト

3.4 HyperTransport

本研究では，評価実験において，アプリケーションのメモリバンド幅ボトルネックを発見するために，PAPI によりメモリアクセス回数を測定する．T2K-Tsukuba の CPU である quad-core Opteron (Barcelona) の L2, L3 キャッシュのライン追い出しアルゴリズムは複雑であり，ソケット上で複数コアが動作している場合の測定が難しい．本研究では PAPI が Opteron の HyperTransport リンク（以下，HT）のトランザクション数をカウントできることに着目し，メモリに接続されている HT のトランザクション数からメモリと CPU の間のデータ転送量を測定する．HT(HyperTransport)[0:3] のうち，どの HT がメモリアクセスとして認識されているかを特定するために，単純なメモリアクセスのみを行うプログラムを使って予備評価を行った．その結果，HT3 の DATA_DWORD_SENT を観測することにより，その CPU ソケットのメモリアクセス回数を取得できることが判明した．

次に，HT3 の DATA_DWORD_SENT の測定値から算出した実効メモリバンド幅の値の正当性の検証を行った．実験環境を表 6 に，結果を図 1 に示す．

図 1 では，どの演算でも PAPI により計測したメモリバンド幅が STREAM ベンチマークの示す値より小さくなっている．この原因については現在調査中だが，全てのメモリアクセスを取得するには観測する HT3 のイベントが DATA_DWORD_SENT のみでは不十分なためと考えられる．よって以上より PAPI での HT3 の観測によるメモリバンド幅測定は現在のところできないことが判明した．以降は PAPI により DCT（DRAM Controller）へのアクセス数をカウントする DRAM_ACCESSES_PAGE を観測し，そこからメモリバンド幅を測定する．

表 6 PAPI による実効メモリバンド幅測定値の正当性の検証：実行環境

ベンチマーク	STREAM Benchmark
使用ノード数	T2K-Tsukuba 1 ノード
MPI プロセス数	1 プロセス
Elements	8000000
numactl オプション	numactl –cpunodebind=0 –localalloc

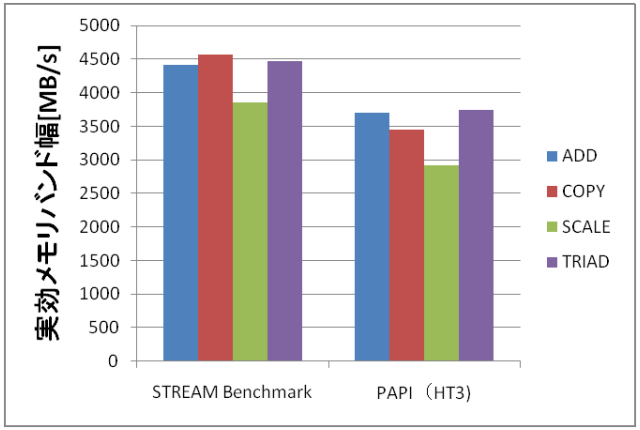

図 1 STREAM Benchmark による測定値と PAPI による測定値の比較

4. 評価結果

本章では評価実験の結果を示す．まず，STREAM Benchmark によってシステムの実効ピークメモリバンド幅を測定し，次に，Lattice QCD において実行プロセス数を増加させた時に性能サチュレーションが起こっているか，実行時間より確認する．また，STREAM Benchmark によって得られたシステムの実効ピークメモリバンド幅と Lattice QCD の測定で得られた実効メモリバンド幅を比較し，性能サチュレーションがメモリバンド幅ボトルネックに起因するものか検証する．

4.1 メモリアクセス性能評価結果

図 2,3 に STREAM Benchmark の実行結果を示す．図 2 には STREAM ベンチマークが示す値，図 3 には PAPI での DRAM_ACCESSES_PAGE の測定結果より求めた値を示

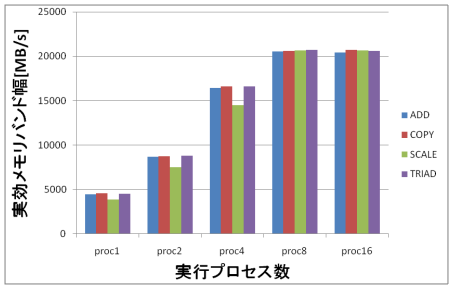

図 2 STREAM Benchmark：実効メモリバンド幅（STREAM ベンチマークの出力）

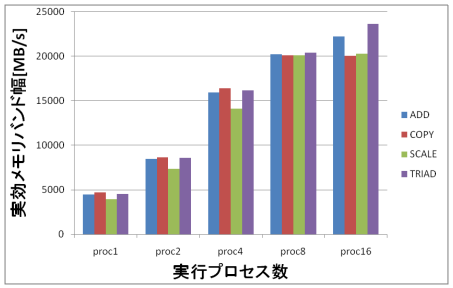

図 3 STREAM Benchmark：実効メモリバンド幅（PAPI による測定結果）

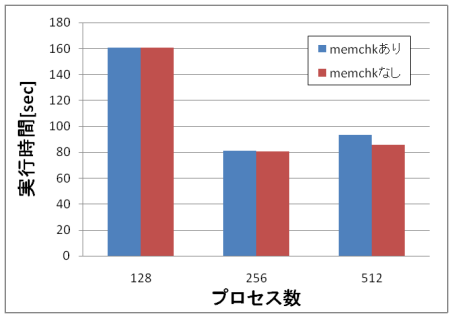

図 4 QCD Benchmark：実行時間

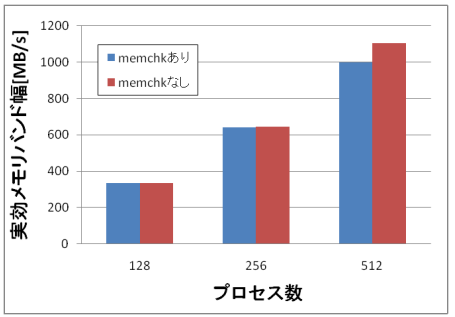

図 5 QCD Benchmark：実効メモリバンド幅

す．両グラフとも横軸は 1 つのノード内で同時実行しているプロセス数，縦軸は実効メモリバンド幅（全プロセスの合計）を示している．図 3 の PAPI で測定した実効メモリバンド幅は，16 プロセス実行時の ADD,TRIAD 演算性能が突出していることを除くと，図 2 の STREAM ベンチマークの出力結果と大部分で同程度であることが読み取れる．これより，PAPI により CPU ソケットからのメモリトラフィックを正しく測定できていることが確認できる．性能が突出している部分に関しては現在調査中である．また，図 2 の 1 プロセス実行時と 2 プロセス実行時の実効メモリバンド幅を比較すると，2 プロセス実行時の性能が 1 プロセス実行時の性能の 2 倍程度になっている．前章で述べたように，NUMA 上でプロセスをソケットに等分割し，かつ全データがホームソケット直結メモリに存在しているため，ソケット数が倍になってメモリバンド幅が倍増していることが観測できる．同様に，4CPU ソケットを使用する 4 プロセス実行時はさらに倍程度になっていることが確認できる．しかし，8 プロセス実行時に注目すると，4 プロセス実行時の 2 倍の性能は出ていないことがわかる．これは，ソケット当たり 1 プロセスの場合は，ソケットに与えられるメモリバンド幅を使い切ってはいないが，2 プロセスにするとその段階でソケットのメモリバンド幅を超えるメモリアクセス要求が発生し，性能が頭打ちになっていると考えられる．また，16 プロセス実行時にはほとんど伸びが見られず，プロセス当たりのメモリバンド幅は大きく低下している．これらの結果から，STREAM ベンチマークのように極端にメモリバンド幅要求が高いプログラムでは，ソケット当たりのメモリバンド幅を少数プロセスで使い切ってしまい，それ以上のプロセス数は無駄であることがわかる．また，T2K-Tsukuba の 1 ノードの最大実効メモリバンド幅は 20GB/s 程度であることが確認できる．[*1]

4.2 QCD ベンチマーク性能評価結果

図 4 に QCD ベンチマークの実行時間を示す．横軸は 1 つのノード内で同時実行しているプロセス数，縦軸は実行時間（全プロセスの合計）を示している．図 5 には PAPI により測定した実効メモリバンド幅を示す．横軸は 1 つのノード内で同時実行しているプロセス数，縦軸は実効メモリバンド幅（全プロセスの合計）を示している．図 4 において memchk プロセスが起動していない場合，4 プロセス実行と比べ 8 プロセス実行では実行時間が半分程度となっており，投入した並列プロセスに見合った性能向上が見られている．しかし，8 プロセス実行と 16 プロセス実行の実行時間はほぼ同じとなっている．これは，前節で述べたように 16 プロセス実行ではメモリアクセス性能が頭打ちとなってしまうため，メモリバンド幅が性能ボトルネックとなっているためであると考えられる．ここで，図 5 の 8 プロセス実行の実効メモリバンド幅を見ると，4 プロセス実行の実効メモリバンド幅の倍程度となっており，性能向上が見られる．しかし，16 プロセス実行を見ると，8 プロセス実行の倍の性能まではいかないものの，ある程度の性能向上が観測される．これは，実効メモリバンド幅では性能向上が見られるが，実行時間では性能向上を確認できないことから，実効メモリバンド幅の測定時に演算のためのメモリアクセスと MPI 通信などの演算以外の処理のためのメモリアクセスが混在しており，それらのメモリアクセスを区別することができないためと

*1 Barcelona Opteron では，NUMA におけるキャッシュコヒーレントチェックのコストが大きく，例え他のソケットのキャッシュにコピーが存在しない場合でも，ノード上の総メモリバンド幅が，各ソケットの単体メモリバンド幅の 4 倍とはならないことが知られている．

考えられる．これらの関係をより明確にするためには，ソケット当たり３プロセス（ノード当たり１２プロセス）の場合を評価すべきであるが，使用した Lattice QCD ベンチマークは問題空間一次元当たりの格子サイズが２の冪乗に制限されており，プロセス空間とノード空間のマッピングを歪めるとノード間通信パターンがノード毎に異なってしまい，メモリバンド幅測定が正確でなくなる恐れがある．よって，今回はノード当たり 4, 8, 16 プロセスのみの実行とした．

また，図 4 の memchk プロセスが起動している場合，8 プロセス実行よりも 16 プロセス実行の実行時間の方が長くなっている．これは，16 プロセス実行時は，ノード内のすべての CPU コアを演算に投入しているため，演算以外のプロセスを実行する場合，演算プロセスが休止状態に入らなければならなく，その結果，休止状態に入った演算プロセスがプログラム全体の遅延を招くためと考えられる．よって，ノード内の 8CPU コアしか演算に使用していない 8 プロセス実行では，演算以外のプロセスを実行する際にも演算処理を休止しないで済むので余剰コアの存在がシステム全体として必要となる daemon 等の不定期なプロセスの実行に回され，性能低下を免れたと推測される．以上で示したように，並列度を上げても性能向上が見られず，メモリバンド幅が性能ボトルネックとなっていると考えられるような状況では，全ての CPU コアを演算に投入することは望ましくないと言える．特に，QCD のように集団通信を多く含んだ並列処理では，並列度が大規模になる程，1 プロセスの処理の遅延が全体の性能低下に顕著に現れるため，やみくもに並列度を上げる事は適切でない．

4.3 関 連 研 究

Torsten Hoefler らによる並列処理における通信専用スレッドについての研究[6] では，通信が必要な並列処理において通信専用スレッドを作成することにより，演算と通信のオーバーラップによる性能向上が可能であることが示されている，ただし，通信専用スレッドが演算の負担とならないようにする必要がある．今回の実験では，実行プロセスと直接関係のないプロセスのみでの検証であったが，演算に使用しない余剰 CPU コアが発生した場合の演算外の処理への CPU コア投入の有効性は十分に期待できる．

5. 考察と今後の課題

HPC アプリケーションといえども，そのメモリバンド幅要求の過多は一概に言えないが，Lattice QCD のように Level3 BLAS のような形でキャッシュヒット率を向上させることができない場合や，流体計算のように長ベクトルに対して演算密度の薄い処理を行う場合，いわゆる Byte/FLOP，すなわち演算当たりに要求されるメモリアクセス要求は高く，性能をキャッシュに依存するスカラプロセッサではベクトルプロセッサに比べ実効性能を上げることが難しい．この傾向は今後，マルチコア化が進むにつれて一層顕著になると思われる．

今回の STREAM ベンチマーク及び Lattice QCD ベンチマークの結果から，マルチコアプロセッサにおける性能がメモリバンド幅をボトルネックとして頭打ちになる現象が起こっていると推測される．これは問題とシステム特性に依存した余剰コアが存在することを意味する．その効果的な利用の一例として，定常的でないシステムプロセスが実行された場合，16core/node としてコアを最大限に利用した場合の性能が 8core/node の場合を下回るという逆転現象が観測された．OS の構成やシステム監視設定によっては，こういった非定常的なシステムプロセスによる割り込みが発生するが，特に大規模並列処理においては，それらがノード毎にずれて発生した場合，個々のノードにおける処理速度低下が僅かであっても，同期処理の妨げとなり，大きく性能を低下させる（いわゆるジッター効果）．実際に，計算科学研究センターの QCD 研究グループでは，今回用いた Lattice QCD ベンチマークとほぼ同様のプログラムを，ノード内の 16 コアを全て用いる形で，最大 128 ノード（2048 プロセス）で実行しているが，memchk daemon の影響は，利用ノード数が多い程大きいという報告がなされている．このような状況において，余剰コアを用いることでシステム性能を低下させない効果が期待できる．

また，従来の並列処理システムにおける通信ライブラリでは，CPU の処理時間を最小化し，ブロック通信におけるレイテンシを低減させることが重要であったが，このような余剰コアを通信補助プロセスに適用し，通信ハードからの割り込み処理や頻度の低い（メモリアクセスを圧迫しない）ポーリング等の処理に割り当て，演算処理リソースはある程度贅沢に利用可能であるというシステム開発やスケジューリングを行うことにより，並列処理全体の性能向上が期待できる．

今回の評価に基づき，今後は様々なアプリケーションに対して，PAPI による実効メモリバンド幅の測定によってメモリバンド幅が性能のボトルネックとなることを定量的に確認していく．また，余剰コアを有効利用するために，アプリケーション毎のメモリバンド幅要求を自動的に測定またはプロファイリングし，必要十分なマルチコアの利用数を求めた後，余剰コア数に応じた補助プロセス生成等を行い，自動最適化を行うツール及びシステムを開発していく予定である．

Vol.2010-HPC-124 No.4
2010/2/22

謝　　辞

本研究の一部は，筑波大学計算科学研究センター・学際共同利用プログラム課題（平成 21 年度後期）「マルチコア／マルチレーン・クラスタにおける並列処理最適化」によるものである．

参 考 文 献

1) 高橋大介, 後藤和茂, 朴泰祐, 建部修見, 佐藤三久, 三上和徳 : T2K 筑波システムにおける Linpack 性能評価, 2008-HPC-116, No. 74, pp. 55em-dash60, 2008.
2) T2K Open Supercomputer Alliance : http://www.open-supercomputer.org/index.html
3) 筑波大学計算科学研究センター : http://www.ccs.tsukuba.ac.jp/CCS/index-j.html
4) The STREAM Benchmark : Computer Memory Bandwidth, http://www.streambench.org/
5) Performance API : http://icl.cs.utk.edu/papi/
6) Torsten Hoefler, Andrew Lumsdaine : Message Progression in Parallel Computing - To Thread or not to Thread?, 2008-Cluster-213, 2008.