Vol.2016-HPC-155 No.21
2016/8/9

コードレベル性能最適化が電力効率に与える影響の分析

今村 智史[1,a)]　安井 雄一郎[1]　稲富 雄一[1]　藤澤 克樹[1]　井上 弘士[1]　小野 貴継[1]

概要：
20 メガワットでのエクサスケール・コンピューティングを実現するためには，HPC システムの電力効率向上が大きな課題となっている．これまでは，アプリケーション実行時の性能向上を目的とし，多くのプログラマがコードレベルでのチューニングに多大な労力を費やしてきた．優れたチューニング技術により性能は劇的に向上してきたものの，現在ではさらに電力効率向上を実現するためのチューニング技術が求められている．しかしながら，コードレベルチューニングが電力効率に与える影響は明らかでない．そこで本研究では，消費電力を考慮せず性能最大化を目標とした従来のコードレベルチューニングが電力効率に与える影響を評価・分析する．代表的な HPC カーネルを実行する 2 種類のプログラムの世界最高性能を達成した実装を用い，それらに適用されたチューニング技術の性能，消費電力，電力効率を実プラットフォームにおいて評価する．また，電力制約下でのアプリケーション実行を想定し，CPU および GPU の動作周波数を変化させた場合に各チューニング技術が電力効率に与える影響を評価する．

1. はじめに

近年，科学技術計算やグラフ解析といった多種多様な HPC アプリケーションが多くの実サービス（たとえば，天気予報やソーシャルネットワークサービス，交通管理等）に使用されている．従来，アプリケーション実行時の性能最大化が HPC 分野における主な目的であったため，多くのプログラマがコードレベルでのチューニングに多大な労力を費やしてきた．実際，Top500 [1] や Graph500 [2] 等の HPC システムの性能を評価するためのベンチマークテストでは，優れたチューニング技術によりベンチマーク・プログラム実行時の性能が劇的に向上してきた．

その一方, 20 メガワットでのエクサスケール・コンピューティングを実現するためには HPC システムの電力効率向上が大きな課題となっており [3], HPC 分野の主要な目標は性能最大化から電力効率最大化へと変化しつつある．そのため，HPC システムの電力効率を評価するためのベンチマークテストも登場し [4], [5]，多くの先行研究において電力効率向上の必要性が主張されている [6], [7], [8], [9], [10], [11]．これらの研究では，ソフトウェアのパラメタや CPU および GPU の動作周波数といったハードウェアのパラメタに対するチューニングが電力効率に与える影響が解析されており，それらを最適化するための手法が提案されている．しかしながら，アルゴリズムやデータ構造を考慮したコードレベルでのチューニングが電力効率に与える影響を明らかにした研究は我々の知る限り未だない．

そこで本論文では，消費電力を一切考慮せず性能最大化を目的とした従来のチューニング手法が性能，消費電力および電力効率に与える影響を評価・分析する．具体的には，Graph500 と SDPARA と呼ばれる 2 種類の HPC プログラムに着目し，これらのプログラムに適用されたチューニング手法を実プラットフォームにおいて評価する．Graph500 は，グラフ解析アプリケーションで広く使用される幅優先探索を実行するプログラムであり，データ参照の局所性が低いためメモリバンド幅が性能に大きく影響する [2]．一方，SDPARA は，データマイニングや機械学習にとって必要不可欠な半正定値計画問題向けの内点法を並列実装したプログラムである [12]．このプログラムは主に密行列に対する浮動小数点演算を行うため，CPU や GPU の性能が重要になる．

これら 2 種類のプログラムは，優れたチューニング手法により世界トップレベルの性能を達成したプログラムであり，かつ，異なる特徴を持つ（メモリバウンドと CPU バウンド）ため，本研究の評価対象として選択した．実機を用いた評価を行った結果，両プログラムに採用されたチューニングにより性能のみならず電力効率も大幅に向上していることが明らかになった．また，電力制約下でのアプリケーション実行を想定し，CPU および GPU の動作周波数を変化させた場合に各チューニング手法が電力効率に

[1] 九州大学
Kyushu University, Japan
[a)] s-imamura@imi.kyushu-u.ac.jp

表 1 実験プラットフォームの仕様

名称	CPU モデル	CPU TDP	CPU 周波数	ノード数	ソケット数×スレッド数	ノード当たりのメモリ容量
SGI UV 300	E7-8890v3	165 W	2.5 GHz	1	16 x 32	16 TB
GPU-cluster*	E5-2640v3	90 W	2.6 GHz	4	2 x 16	256 GB

*GPU-cluster はノード毎に 2 つの Tesla K40m GPU（ベース周波数：745 MHz）を搭載.

与える影響を評価する．メモリ性能に着目したチューニングが適用された Graph500 に関しては，動作周波数の変化に伴い電力効率の向上比が大きく変化し，かつ，適用されたチューニングによってその変化の傾向が異なることが分かった．一方，メモリ性能に着目したチューニングが適用されていない SDPARA に関しては，動作周波数の変化に関わらず電力効率の向上比がほぼ一定であることが明らかになった．

2. 実験環境

本節では，評価実験に使用するプラットフォームの仕様と 2 種類の HPC プログラムの概要を説明する．

2.1 プラットフォーム

本研究の評価実験では，**表 1** に示した 2 種類の実プラットフォームを使用する．SGI UV 300 は 16 ソケットを搭載した単一ノードシステムであり，Graph500 の評価に用いる．また，GPU-cluster はノード毎に CPU と GPU をそれぞれ 2 つずつ搭載した 4 ノードシステムであり，SDPARA の評価に使用する．両プラットフォームはモデルの異なる Intel Xeon Haswell プロセッサを搭載しており，Intel Hyper-Threading を適用している．特に記述がない限り，実験には CPU と GPU ともに表に示したベース周波数を使用する．

消費電力の測定には，Intel の RAPL（Running Average Power Limit）を使用する．これは，Intel Sandy Bridge アーキテクチャから導入されたモデルベースの電力管理インターフェイスであり，CPU パッケージと DRAM の電力測定をサポートしている [13]．本論文に記載する消費電力の値はシステム全体の消費電力ではなく，CPU と DRAM（GPU-cluster では加えて GPU）の消費電力の合計であることに注意されたい．そのため，システム全体の消費電力に比べ値は小さいが，その消費電力と RAPL により計測した消費電力は強い相関関係にあることが先行研究によって示されている [14]．したがって，RAPL を用いて 取得した消費電力およびその値を基に算出する電力効率の比較は正確に行うことができ る．なお，GPU の消費電力は内蔵の電力センサを用いて計測する．

2.2 HPC プログラム

2.2.1 Graph500

Graph500 はデータインテンシブなアプリケーションに対する HPC システムの性能を評価するために開発されたベンチマーク・プログラムである [2]．このプログラムは，*Generation*，*Construction*，*BFS* と呼ばれる 3 つのステップを実行する．Generation ステップでは，R-MAT（recursive matrix）演算 [15] を使用して Kronecker グラフ [16] の枝リストを生成する．頂点数の 2 の対数である *scale* と頂点数に対する枝数の比率である *edgefactor* の 2 つのパラメタを入力として受け取り，2^{scale} 個の頂点と $2^{scale} \cdot edgefactor$ 本の枝を出力する．したがって，プログラムの実行中に消費されるメモリ容量は scale が 1 増加する毎に倍になる．次に Construction ステップでは，Generation ステップで得られた枝リストからグラフを生成する．そして，BFS ステップでは，それぞれ異なる頂点からの幅優先探索とその出力結果の検証を 64 回繰り返し行う．各ステップの処理は OpenMP を用いて並列化されている．このプログラムの性能は，BFS ステップにおいて 1 秒当たりに探索した枝数 TEPS（Traversed Edges Per Second）を用いて評価される．したがって，電力効率はその値を平均消費電力で割った TEPS/Watt となる．

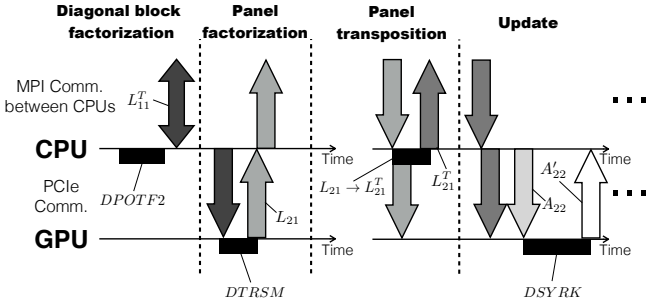


図 1 SDPARA Cholesky の処理内容の概略図

2.2.2 SDPARA

SDPARA は，半正定値計画問題 SDP（Semidefinite Programming）用の内定法を並列実装したプログラムである [12]．SDP は数理最適化の中で代表的な問題であり，組み合わせ最適化や制御理論，データマイニング，機械学習等幅広い分野において研究がなされている．このプログラムは *Elements* と *Cholesky* と呼ばれる 2 つのコンポーネントに分けられる．Elements は Schur complement matrix（SCM）の計算を行う部分であり，メモリバウンドである．これに対し，Cholesky は SCM の Cholesky factorization を行うため，CPU バウンドである．メモリバウンドな Graph500 とは異なり CPU バウンドな特徴を持つプログラムの評価を SDPARA を用いて行うため，本論文では後者の Cholesky にのみ着目する．

Cholesky では，$N \times N$ の行列 A を下三角行列 L とその転置行列 L^T に分解する Cholesky factorization を行う．行列 A は $n_b \times n_b$ の小さなブロックに分割され，複数の MPI プロセスに分配される．各プロセスはそれぞれのブロックに対し *Diagonal block factorization*，*Panel factorization*，*Panel transposition*，*Update* と呼ばれる 4 つのステップを CPU と GPU においてそれぞれ実行する．各処理には ScaLAPACK ライブラリ [17], [18] で提供されるルーチンが使用される．また，各プロセスの処理は OpenBLAS 0.2.14 [19], [20] と CUBLAS 7.5 [21] を用いて並列化されている．**図 1** を用いて各ステップの概要を説明する（詳細については [17], [18] を参照されたい）．図中の矢印はプロセス間の MPI データ通信もしくは CPU-GPU 間の PCIe データ通信を表しており，黒い四角は CPU および GPU における演算処理を表している．

(1) *Diagonal block factorization*: ある 1 プロセスが CPU において DPOTF2 ルーチンを実行し，$n_b \times n_b$ のブロック A_{11} を下三角行列 L_{11} とその転置行列 L_{11}^T に分解する．その後，L_{11}^T を他のプロセスへブロードキャストする．

(2) *Panel factorization*: L_{11}^T を受け取った各プロセスは PCIe を介してそのデータを GPU へ転送し，GPU において DTRSM ルーチンを実行し L_{21} を算出する．その結果は CPU へ転送され，他のプロセスへブロードキャストされる．

(3) *Panel transposition*: L_{21} を受け取った各プロセスは CPU において転置行列 L_{21}^T を算出し，その結果を他のプロセスへブロードキャストする．また，その処理と並行して L_{21} を GPU へ転送する．

(4) *Update*: 各プロセスは L_{21}^T を受け取り，そのデータを各プロセスが持つ A_{22} と共に GPU へ転送する．その後，GPU における DSYRK ルーチンの実行により A'_{22} を算出し，その結果を CPU に転送する．

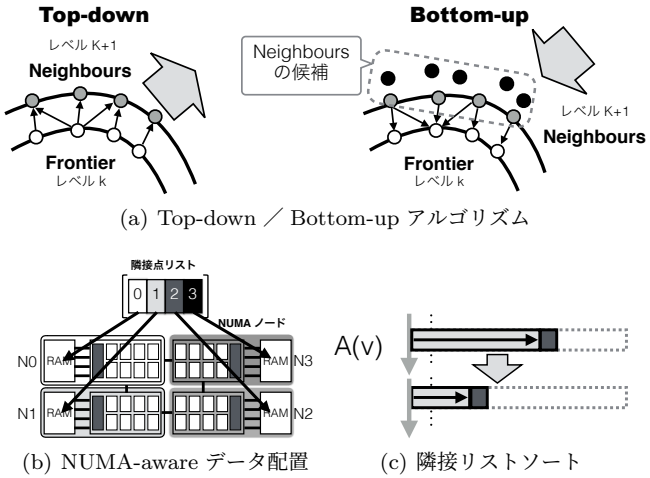
$(n - n_b) \times (n - n_b)$ の行列 A'_{22} に対して上記 4 ステップを再帰的に実行することで Cholesky factorization が行われる．SDPARA Cholesky の性能は 1 秒当たりに実行される浮動小数点演算数 FLOPS（FLOps Per Second）を用いて評価する．電力効率の指標はその値を平均消費電力で割った FLOPS/Watt である．

3. 性能チューニングが電力効率に与える影響の評価

本節では，まず，Graph500 と SDPARA それぞれに対し性能向上を目的として適用されたコードレベルチューニング手法を説明する．次に，それらのチューニングが性能，平均／ピーク消費電力，電力効率（電力当たりの性能）に与える影響の評価を行う．

表 2 3 種類の Graph500 実装の比較

チューニング	実装		
	Reference [2]	Yasui13 [22]	Yasui16 [23]
Top-down [2]	✓		
Direction opt. [24]		✓	✓
NUMA データ配置 [22]		✓	✓
隣接点リストソート [25]			✓
頂点ソート [26], [27]			✓



(a) Top-down ／ Bottom-up アルゴリズム

(b) NUMA-aware データ配置　(c) 隣接リストソート

図 2 Graph500 に適用されたチューニング手法

3.1 Graph500 に対するチューニング手法

本論文では，**表 2** に示した 3 種類の Graph500 実装（Reference，Yasui13，Yasui16）の比較を行う．Reference は**図 2(a)** の左図に示す Top-down（Level-synchronized）アルゴリズムを適用している．このアルゴリズムでは，あるルート頂点から幅優先探索を開始し，隣接した（枝が繋がっている）子頂点を同心円上に探索する．各頂点に対する隣接点は隣接点リストと呼ばれるデータ構造に格納され，隣接点を探索する際にはこのデータ構造が走査される．同一世代の頂点集合をレベルと呼び，各レベルにおける探索は複数のスレッドで並列に行われる．隣接した子頂点が未探索の場合，その親頂点を *predecessor map* と呼ばれる木データ構造に追加し，探索を終了した子頂点を *visited*（探索済み）状態にする．複数スレッドが同時に同じ頂点を探索する可能性があるため，頂点が visited 状態か否かのチェックはアトミックに行う必要がある．最も世代の新しい頂点を含む外側のレベルは *frontier* と呼ばれ，frontier 上の全ての頂点に接続された次のレベル（*neighbours*）の子頂点を繰り返し探索する．このアルゴリズムは frontier から出る全ての枝を探索する必要があるが，接続していない頂点を探索する必要がない．そのため，少数の頂点を含む frontier に対し高速な探索ができる．

Yasui13 [22] は，不必要な枝の探索を削減するために，上記の Top-down アルゴリズムと 図 2(a) の右図に示した Bottom-up アルゴリズムをレベル毎に切り替える *direction*

optimizing [24] を適用している．Bottom-up アルゴリズムは，neighbours に含まれる頂点の候補となる frontier の外側の未探索頂点に対し，各頂点の隣接点リストから frontier に含まれる親頂点を探索する．このアルゴリズムでは，全ての未探索頂点を探索する必要があるが，親頂点を見つけた後の枝探索を省くことができる．したがって，多くの頂点を含む frontier に対して効率が良い．さらに，未探索頂点に対する探索のみを行うため，Top-down アルゴリズムとは異なり頂点が未探索であるか否かをチェックするためのアトミック処理を必要としない．また，Yasui13 には，NUMA（Non-Uniform Memory Access）アーキテクチャにおけるリモートメモリアクセスを削減するためのデータ配置が適用されている．**図 2(b)** に示すように，隣接点リストが NUMA ノード数と等しい数に分割され，各ノードにそれぞれ配置される．これにより，各ノード上のスレッドはローカルノードに配置された隣接点のみを独立して探索できる．以上のチューニング手法により，この実装は第 4 回 Graph500 リスト（2012 年 6 月更新）において，シングルノードシステムの中で世界最高性能を達成している [28].

Yasui16 [23] では，上記のチューニング手法に加え，隣接点リストソート [25] と頂点ソート [26], [27] が適用されている．隣接点リストソートとは，隣接点リスト内の頂点を次数の降順にソートする手法である．Bottom-up アルゴリズムでは，各未探索頂点の隣接点リストを frontier 上の親頂点が見つかるまで走査する．次数の高い（より多くの枝を持つ）頂点は親頂点となる可能性がより高いため，このソートにより**図 2(c)** のようにリストの走査を早い段階で終了できる場合が多くなる．また，頂点ソートとは，配列に格納される頂点データそのものを次数の降順にソートする手法である．これにより，次数の近い頂点同士を物理メモリ上の近い位置に配置できる．次数の高い頂点データほど頻繁にアクセスされるため，そのアクセスによってアクセス頻度の低い頂点データがキャッシュメモリから追い出されることを防ぐことができ，キャッシュメモリのヒット率が向上できる．Yasui16 は，第 12 回 Graph500 リスト（2016 年 6 月更新）において，シングルノードシステムの中で世界最高性能を達成している [28].

3.2 Graph500 に対するチューニングの電力効率評価

上記 3 種類の Graph500 実装（Reference，Yasui13，Yasui16）を SGI UV 300 において実行し，性能（TEPS），平均／ピーク消費電力，電力効率（TEPS/Watt）を計測した結果を**表 3** にまとめる．表内の数値は Reference に対する相対値であり，カッコ内の数値は Yasui13 に対する相対値を表している．平均／ピーク電力は 16 ソケットの CPU と DRAM の消費電力を合計したものである．また，詳細な分析のために，探索枝数とローカルメモリアクセス率（総メモリアクセス回数に対するローカルメモリアクセスの割合），1000 命令毎のラストレベルキャッシュメモリのミス数（LLC MPKI）を記載している．ローカルメモリアクセス率と LLC MPKI はハードウェアパフォーマンス・カウンタから取得した．

表 3 3 種類の Graph500 実装の評価結果

	実装		
評価指標	Reference	Yasui13	Yasui16
TEPS	x1.00	x156.36	x532.61 (x3.41)
平均電力	x1.00	x1.13	x1.26 (x1.11)
ピーク電力	x1.00	x1.07	x1.11 (x1.03)
TEPS/Watt	x1.00	x137.80	x423.61 (x3.07)
探索枝数	8.59E+9	4.19E+8	3.36E+8
ローカル率	0.25	0.99	0.98
LLC MPKI	7.72	3.61	1.69

スレッド数：512，scale：29，プラットフォーム：SGI UV 300

Yasui13 は Reference に比べ，156 倍の TEPS を達成したにも関わらず，平均電力の増加とピーク電力の増加はそれぞれわずか 13%と 7%であった．その結果，TEPS/Watt は 138 倍向上している．Direction optimizing により Top-down アルゴリズムに比べ探索枝数が 95%削減されており，これが性能向上の大きな要因である．探索枝数の削減により消費電力が増加することはないため，このチューニングは電力効率向上に極めて効果的である．消費電力が増加した理由は，Direction optimizaing で選択される Bottom-up アルゴリズムにより Top-down アルゴリズムでは必要なアトミック処理が排除され，CPU 使用率が上昇したためである．また，Yasui13 は LLC MPKI を 7.72 から 3.61 に削減している．Top-down アルゴリズムは frontier に含まれる頂点のみを辿るため，ランダムなメモリアクセスパターンを持つ．これに対し，Bottom-up アルゴリズムは全ての未探索頂点を辿るため，規則的なメモリアクセスパターンを持ちキャッシュヒット率が高くなる．さらに，NUMA-aware なデータ配置によって 99%のメモリアクセスがローカルメモリへのアクセスであり，これも性能向上に大きく貢献している．

Yasui16 は Rerefernce に対し，533 倍の TEPS 向上にも関わらず，平均電力の増加とピーク電力の増加はそれぞれ 11%と 3%であった．Yasui13 と比較すると，11%の平均電力増加と 3%のピーク電力増加，3.4 倍の TEPS 向上である．電力効率は Reference に対し 424 倍，Yasui13 に対し 3 倍向上した．隣接点リストソートにより Yasui13 に比べ探索枝数が 20% 削減され，頂点ソートにより LLC MPKI が 3.61 から 1.69 に減少している．消費電力増加の理由は，LLC MPKI の減少によりメモリストール時間が短縮され，CPU の稼働率が増加したためだと考えられる．

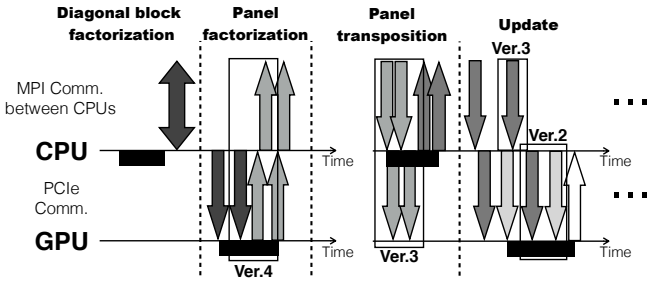


図 3 SDPARA Cholesky に対するオーバーラップチューニング

表 4 SDPARA Cholesky の 4 種類の実装の比較

オーバーラップ	Ver.1 [17]	Ver.2 [17]	Ver.3 [17]	Ver.4 [18]
L_{21}^T，A_{22} の PCIe 通信と DSYRK（update）		✓	✓	✓
L_{21} の MPI 通信と PCIe 通信（panel tran.）			✓	✓
L_{21}^T の MPI 通信と DSYRK（update）			✓	✓
L_{11}^T，L_{21} の PCIe 通信と DTRSM（panel fact.）				✓

3.3 SDPARA Cholesky に対するチューニング手法

SDPARA Cholesky に対しては，4 種類のバージョンの比較を行う．ベース実装（Ver.1）は，図 1 に示した diagonal block factorization から Update までの 4 ステップに含まれる演算処理とデータ通信を逐次的に行う．これに対し，他の 3 つのバージョン（Ver.2-4）では**図 3** に示す演算処理とデータ通信のオーバーラップを行っている．**表 4** に各バージョンに適用されたオーバーラップ手法をまとめる．Ver.2 では，update ステップにおいて，L_{21}^T と A_{22} データの PCIe 通信と GPU における DSYRK ルーチンの実行をオーバーラップさせる．それぞれのデータを小さなブロックに分割し，各ブロックに対して DSYRK ルーチンを実行することでオーバーラップが可能となる．Ver.3 ではこれに加え，update ステップにおける L_{21}^T データの MPI 通信と DSYRK ルーチンの実行をオーバーラップする．さらに，panel transposition ステップにおいて，L_{21} データのプロセス間 MPI 通信と当該データの PCIe 通信をオーバーラップさせる．小さなブロックに分割された L_{21} データがそれぞれブロードキャストされ，各プロセスはそのブロックを受け取り次第 GPU へ転送する．最後に Ver.4 では，上記のオーバーラップ手法に加え，panel factorization ステップにおいて L_{11}^T と L_{21} データの PCIe 通信と GPU における DTRSM ルーチンの実行をオーバーラップさせる．各プロセスは L_{11}^T の部分ブロックを順次 GPU へ転送し，DTRSM ルーチンの実行を開始する．各 DTRSM の実行が終了し次第，L_{21} の部分ブロックを CPU に転送する．

表 5 SDPARA Cholesky の 4 種類の実装の評価

指標	実装 Ver.1	Ver.2	Ver.3	Ver.4
FLOPS	x1.00	x1.18	x1.23 (x1.04)	x1.31 (x1.07)
平均電力	x1.00	x1.00	x1.01 (x1.01)	x1.04 (x1.03)
ピーク電力	x1.00	x1.01	x0.99 (x0.98)	x0.96 (x0.96)
FLOPS/Watt	x1.00	x1.19	x1.22 (x1.02)	x1.26 (x1.03)

行列サイズ N：107,206，ブロックサイズ nb：1,024，プラットフォーム：GPU-cluster

3.4 SDPARA Cholesky に対するチューニングの電力効率評価

表 5 に GPU-cluster において計測した 4 種類の実装の性能（FLOPS），平均電力とピーク電力，電力効率（FLOPS/Watt）を示す．表中の数値は Ver.1 の値で正規化したものであり，カッコ内の数値はバージョンが 1 つ低い実装に対する相対値を表している．消費電力は 4 ノードの CPU，GPU，DRAM の消費電力の合計値である（各ノードは CPU と GPU を 2 つずつ搭載）．行列サイズ N とブロックサイズ nb は事前実験によりそれぞれ 107,206 と 1,024 に設定した．

表 5 に示した結果から，いずれのオーバーラップ手法でも性能が向上していることが分かる．特に，4 つのステップの内 update ステップの実行時間が最も長いため，Ver.2 で適用された update ステップにおけるオーバーラップが性能を大きく向上している．全オーバーラップ手法を適用した Ver.4 では，Ver.1 に対し 31%性能が向上した．これらのオーバーラップ手法は CPU と GPU を同時に稼働させるため，直感的にはオーバーラップを行わない場合に比べ消費電力が増加することが予想できる．しかしながら，平均電力の増加はわずか 4%であった．Cholesky 実行時の時間経過に伴う CPU，GPU，DRAM の消費電力の推移をそれぞれ調査したところ，オーバーラップを適用しない Ver.1 においても CPU が常に稼働していることが分かった．これは CPU で定期的に実行される CUDA API の実装によるものである．結果的に，オーバーラップ手法を適用することでわずかな消費電力増加により性能が向上したため，電力効率も向上した．

また，Ver.4 では他の実装に比べ性能が向上しているにも関わらずピーク消費電力が減少した．Ver.3 と 4 の実装の違いは，panel factorization ステップにおける GPU の演算処理と PCIe 通信のオーバーラップである．Ver.3 では L_{11}^T データ全体を一括して CPU から GPU へ転送し，DTRSM ルーチンの実行により L_{21} を算出し，その結果を一括して CPU に送り返す．これに対し Ver.4 では，複数の小さなブロックに分割した L_{11}^T データに対し，同様の処理を行う．GPU での演算に対する入力データサイズが Ver.4 では小さくなるため，GPU の使用率が低下しピーク消費電力が低下したと考えられる．

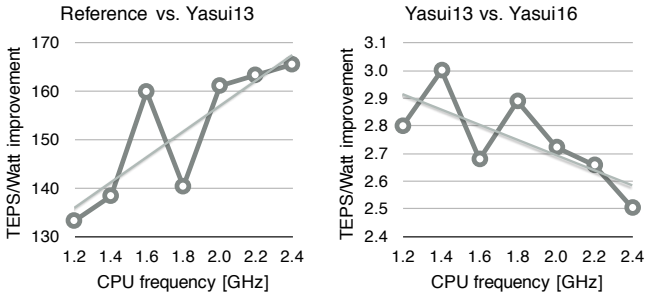


図 4 動作周波数の変化に対するチューニングによる電力効率向上比の変化（Graph500）

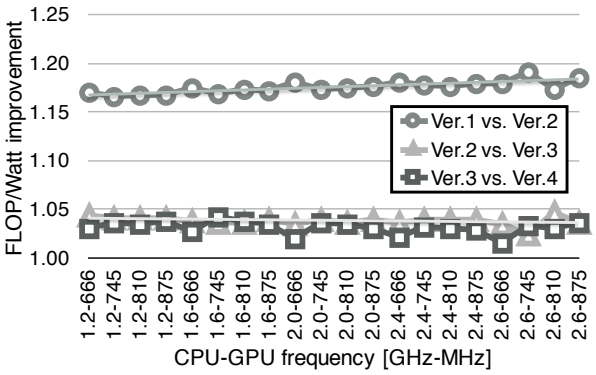


図 5 動作周波数の変化に対するチューニングによる電力効率向上比の変化（SDPARA Cholesky）

4. 電力制約を想定した場合に性能チューニングが電力効率に与える影響の評価

本節では，プラットフォームに電力制約が設定される場合を想定し，Graph500 と SDPARA に適用されたチューニング手法が電力効率に与える影響の評価を行う．現在のサーバプラットフォームには，プラットフォーム全体の消費電力がユーザが設定した電力制約値を超過しないよう管理する機能が搭載されている．さらに，データセンタやスーパーコンピュータの演算性能を向上するための Over-provisioning システムが近年注目されており，電力制約下での効率的なアプリケーション実行が重要となっている [29]．一般的に，電力制約を満たすためには CPU および GPU の動作周波数が動的に制御される．そこで，CPU と GPU の動作周波数を変化させた場合にチューニング手法が電力効率に与える影響の変化を評価する．

4.1 Graph500

SGI UV 300 において CPU 動作周波数を変化させた場合に，Graph500 に対するチューニングが電力効率に及ぼす影響を**図 4** に示す．図中の線は回帰直線であり，CPU の動作周波数は 1.2 GHz から 2.4 GHz まで 0.2 GHz 刻みで変化させた．図 4 の左図は，Reference に対する Yasui13 の電力効率向上比を示している．CPU 動作周波数が 1.6 GHz と 1.8 GHz の場合には電力効率の向上比が大きく増減しているものの，全体として動作周波数が上昇するにつれて電力効率向上比が大きくなる傾向にある．これに対し，右図は Yasui13 に対する Yasui16 の電力効率向上比を示しており，動作周波数が上昇するにつれて電力効率向上比が小さくなることが分かる．Graph500 実行時の性能はメモリの性能に大きく影響されるため，NUMA アーキテクチャにおいてリモートメモリアクセスを削減するためのデータ配置やキャッシュミス数を削減するための頂点ソートといったメモリ性能に着目したチューニングが Graph500 には適用されている．CPU の動作周波数が変化すると，CPU の性能とメモリの性能のバランスが変化するため，図 4 のような傾向が見られたと考えられる．左図と右図の傾向の違いの分析は今後の課題とする．

4.2 SDPARA Cholesky

GPU-cluster において CPU と GPU の動作周波数を変化させた場合に，SDPARA に対するチューニングが電力効率に及ぼす影響を**図 5** に示す．各線はそれぞれの実装間での電力効率向上比を表している．横軸は CPU と GPU の動作周波数の組み合わせを示しており，たとえば，1.2-666 は CPU の動作周波数が 1.2 GHz であり，GPU の動作周波数が 666 MHz であることを意味する．図 5 では，Graph500 の結果と異なり，チューニングによる電力効率向上比が動作周波数の変化に関わらずほぼ一定であることが分かる．これは，SDPARA Cholesky が CPU バウンドであり，このプログラムに適用されたチューニングが CPU と GPU のメモリ性能に影響しないためである．

5. 関連研究

5.1 BFS と SDPARA に対する性能チューニング手法

BFS はグラフ解析アプリケーションにとって重要なカーネルであるため，シングルノードシステムおよび大規模分散システムを対象とした数多くのチューニングがなされてきた．Yoo らはノード間での通信量を削減するために，従来の 1 次元分割（頂点分割）に代わる 2 次元分割（枝分割）手法を提案している [30]．また，ノード間の通信量を削減するためにデータ圧縮技術も使用される [31], [32]．Checconi らは分散システムにおけるスレッド間およびノード間でのロードバランスを保つ手法を提案している [33]．また，Ueno らは GPU 向けの BFS アルゴリズムを実装し，GPU 間の通信量をデータ圧縮により削減した [34]．Bader と Madduri は，Graph500 の Reference 実装に採用されている Level-synchronized（Top-down）アルゴリズムを実装した [35]． さらに，探索枝数を削減するために Top-down アルゴリズムと Bottom-up アルゴリズムを切り替える Direction optimizing が Beamer らによって提案され，BFS の 性能が大幅に向上した [24]．近年では，メモリ

システムに着目したチューニング手法も提案されている．例えば，データ局所性の向上によりキャッシュミス数を削減するためのデータ配置 [36] や頂点ソート [26], [27] 等が挙げられる．また，Frasca らは BFS 実行時のエネルギー効率を考慮し，NUMA システムにおけるデータ配置の最適化を行った [37].

SDPARA に含まれる Cholesky factorization もまた多様な分野において応用される重要なカーネルであり，多くのチューニング手法により性能が向上してきた．Fujisawa らは高いデータ並列性を活用するために CPU と GPU で協調実行可能なアルゴリズムを実装した [17]．この実装には，3.3 節に示したオーバーラップ手法が適用されている．彼らはまた，スレッドの CPU アフィニティ設定とメモリインターリーブの適用により性能向上を達成している [18]．Tsujita らはスケーラブルなデータ転送方式と termination detection 手法によりスケーラビリティを改善した．彼らの実装は，TSUBAME2.5 スーパーコンピュータに搭載された 4,080GPU において 1.774 PFLOS の性能を達成している [38]． 本論文では，3 節で述べたように，世界最高性能を達成した Graph500 に対する性能チューニング手法 [22], [25], [27] と SDPARA Cholesky に対する性能チューニング手法が電力効率に与える影響を定量的に評価した．

5.2 電力効率向上のための HW/SW パラメタチューニング手法

CPU の動作周波数は電力効率に大きく影響するパラメタであるため，MPI プロセス間のロードインバランス [6] やループイタレーション毎のプログラムの振る舞い [8] に着目した動作周波数チューニングがなされている．また，いくつかの先行研究では，ソフトウェア／ハードウェアのパラメタチューニングが性能，消費電力，消費エネルギーに与える影響を包括的に調査している [7], [9], [10], [11]．これらの論文では，チューニングパラメタとして，CPU の動作周波数に加え，ループタイリングやループアンローリング，スレッド数とノード数が考慮されており，性能，消費電力，電力効率のそれぞれを最適化するための自動チューニングツールが提案されている．これらの先行研究がハードウェアとソフトウェアのパラメタチューニングを行っているのに対し，本論文ではアルゴリズムとデータ構造を考慮したコードレベル性能チューニング手法が性能，消費電力，電力効率に与える影響の調査を行った．

6. おわりに

本論文では，Graph500 と SDPARA の 2 種類のプログラムに着目し，それらに適用された性能チューニングが性能，消費電力，電力効率に与える影響を定量的に評価した．評価実験の結果，これらのプログラムに適用されたチューニング手法は性能のみならず電力効率も大幅に向上していることが明らかになった．また，動作周波数の変化に対するチューニングの電力効率への影響を評価した結果，メモリバウンドである Graph500 では，チューニングの種類によって電力効率向上比の変化が大きく異なることが分かった．今後の予定として，各チューニング手法をより細かく切り分けて実装し，それぞれが電力効率に与える影響を詳細に評価する．

謝辞 本研究は，一部，JST CREST「ポストペタスケール高性能計算に資するシステムソフトウェア技術の創出」による．

参考文献

[1] Dongarra, J.: *Performance of Various Computers Using Standard Linear Equations Software*, University of Tennessee, Knoxville TN, 37996 (2014).
[2] Murphy, R. C., Wheeler, K. B., Barrett, B. W. and Ang, J. A.: *Introducing the Graph 500*, Cray User's Group (CUG) (2010).
[3] Bergman, K., Borkar, S., Campbell, D., Carlson, W., Dally, W., Denneau, M., Franzon, P., Harrod, W., Hiller, J., Karp, S., Keckler, S., Klein, D., Lucas, R., Richards, M., Scarpelli, A., Scott, S., Snavely, A., Sterling, T., Williams, R. S., Yelick, K., Bergman, K., Borkar, S., Campbell, D., Carlson, W., Dally, W., Denneau, M., Franzon, P., Harrod, W., Hiller, J., Keckler, S., Klein, D., Kogge, P., Williams, R. S. and Yelick, K.: ExaScale Computing Study: Technology Challenges in Achieving Exascale Systems Peter Kogge, Editor & Study Lead (2008).
[4] Subramaniam, B. and Feng, W.: Towards Energy-Proportional Computing for Enterprise-Class Server Workloads, ICPE '13 (2013).
[5] Hoefler, T.: Green Graph 500, `http://green.graph500.org/index.php`.
[6] Kappiah, N., Freeh, V. W. and Lowenthal, D. K.: Just In Time Dynamic Voltage Scaling: Exploiting Inter-Node Slack to Save Energy in MPI Programs, SC '05, pp. 33–41 (2005).
[7] Tiwari, A., Laurenzano, M. A., Carrington, L. and Snavely, A.: Auto-tuning for Energy Usage in Scientific Applications, Euro-Par'11, pp. 178–187 (2011).
[8] Laurenzano, M. A., Meswani, M., Carrington, L., Snavely, A., Tikir, M. M. and Poole, S.: Reducing Energy Usage with Memory and Computation-aware Dynamic Frequency Scaling, Euro-Par'11, Berlin, Heidelberg, pp. 79–90 (2011).
[9] Miceli, R., Civario, G., Sikora, A., César, E., Gerndt, M., Haitof, H., Navarrete, C., Benkner, S., Sandrieser, M., Morin, L. and Bodin, F.: AutoTune: A Plugin-driven Approach to the Automatic Tuning of Parallel Applications, PARA'13, pp. 328–342 (2013).
[10] Balaprakash, P., Tiwari, A. and Wild, S. M.: Multi-Objective Optimization of HPC Kernels for Performance, Power, and Energy, *High Performance Computing Systems. Performance Modeling, Benchmarking and Simulation*, Springer, pp. 239–260 (2013).
[11] Schöne, R., Treibig, J., Dolz, M. F., Guillen, C., Navarrete, C., Knobloch, M. and Rountree, B.: Tools and Methods for Measuring and Tuning the Energy Efficiency

of HPC Systems, *Scientific Programming*, Vol. 22, No. 4, pp. 273–283 (2014).

[12] Yamashita, M., Fujisawa, K. and Kojima, M.: SDPARA: SemiDefinite Programming Algorithm paRAllel Version, *Parallel Comput.*, Vol. 29, No. 8, pp. 1053–1067 (2003).

[13] David, H., Gorbatov, E., Hanebutte, U. R., Khanna, R. and Le, C.: RAPL: Memory Power Estimation and Capping, ISLPED '10, pp. 189–194 (2010).

[14] Hackenberg, D., Ilsche, T., Schone, R., Molka, D., Schmidt, M. and Nagel, W. E.: Power Measurement Techniques on Standard Compute Nodes: A Quantitative Comparison, ISPASS '13, pp. 194–204 (2013).

[15] Deepayan, C., Yiping, Z. and Christos, F.: R-MAT: A Recursive Model for Graph Mining, *Proceedings of the 4th SIAM International Conference on Data Mining*, pp. 442–446 (2004).

[16] Leskovec, J., Chakrabarti, D., Kleinberg, J., Faloutsos, C. and Ghahramani, Z.: Kronecker Graphs: An Approach to Modeling Networks, *J. Mach. Learn. Res.*, Vol. 11, pp. 985–1042 (2010).

[17] Fujisawa, K., Sato, H., Matsuoka, S., Endo, T., Yamashita, M. and Nakata, M.: High-Performance General Solver for Extremely Large-Scale Semidefinite Programming Problems, SC '12, pp. 93:1–93:11 (2012).

[18] Fujisawa, K., Endo, T., Yasui, Y., Sato, H., Matsuzawa, N., Matsuoka, S. and Waki, H.: Peta-scale General Solver for Semidefinite Programming Problems with over Two Million Constraints, IPDPS '14, pp. 1171–1180 (2014).

[19] Xianyi, Z., Qian, W. and Yunquan, Z.: Model-driven Level 3 BLAS Performance Optimization on Loongson 3A Processor, ICPADS '12, pp. 684–691 (2012).

[20] Wang, Q., Zhang, X., Zhang, Y. and Yi, Q.: AUGEM: Automatically Generate High Performance Dense Linear Algebra Kernels on x86 CPUs, SC '13, pp. 25:1–25:12 (2013).

[21] NVIDIA: CUBLAS LIBRARY User Guide DU-06702-001_v7.5 (2015).

[22] Yasui, Y., Fujisawa, K. and Goto, K.: NUMA-optimized Parallel Breadth-first Search on Multicore Single-node System, Big Data '13, pp. 394–402 (2013).

[23] Yasui, Y., Fujisawa, K., Goh, E. L., Baron, J., Sugiura, A. and Uchiyama, T.: NUMA-aware Scalable Graph Traversal on SGI UV Systems, HPGP '16, pp. 19–26 (2016).

[24] Beamer, S., Asanović, K. and Patterson, D.: Direction-Optimizing Breadth-first Search, SC '12, pp. 12:1–12:10 (2012).

[25] Yasui, Y., Fujisawa, K. and Sato, Y.: Fast and Energy-efficient Breadth-First Search on a Single NUMA System, ISC '14, pp. 365–381 (2014).

[26] Ueno, K. and Suzumura, T.: Highly Scalable Graph Search for the Graph500 Benchmark, HPDC '12, pp. 149–160 (2012).

[27] Yasui, Y. and Fujisawa, K.: Fast and Scalable NUMA-based Thread Parallel Breadth-first Search, HPCS '15, pp. 377–385 (2015).

[28] Graph500: The Graph 500 list, `http://www.graph500.org`.

[29] Inadomi, Y., Patki, T., Inoue, K., Aoyagi, M., Rountree, B., Schulz, M., Lowenthal, D., Wada, Y., Fukazawa, K., Ueda, M., Kondo, M. and Miyoshi, I.: Analyzing and Mitigating the Impact of Manufacturing Variability in Power-Constrained Supercomputing, SC '15, pp. 78:1–78:12 (2015).

[30] Yoo, A., Chow, E., Henderson, K., McLendon, W., Hendrickson, B. and Catalyurek, U.: A Scalable Distributed Parallel Breadth-First Search Algorithm on BlueGene/L, SC '05, pp. 25–43 (2005).

[31] Checconi, F., Petrini, F., Willcock, J., Lumsdaine, A., Choudhury, A. R. and Sabharwal, Y.: Breaking the Speed and Scalability Barriers for Graph Exploration on Distributed-memory Machines, SC '12, pp. 13:1–13:12 (2012).

[32] Satish, N., Kim, C., Chhugani, J. and Dubey, P.: Large-Scale Energy-efficient Graph Traversal: A Path to Efficient Data-intensive Supercomputing, SC '12, pp. 14:1–14:11 (2012).

[33] Checconi, F. and Petrini, F.: Traversing Trillions of Edges in Real Time: Graph Exploration on Large-Scale Parallel Machines, IPDPS '14, pp. 425–434 (2014).

[34] Ueno, K. and Suzumura, T.: Parallel Distributed Breadth First Search on GPU, HiPC '13, pp. 314–323 (2013).

[35] Bader, D. A. and Madduri, K.: Designing Multithreaded Algorithms for Breadth-First Search and St-connectivity on the Cray MTA-2, ICPP '06, pp. 523–530 (2006).

[36] Agarwal, V., Petrini, F., Pasetto, D. and Bader, D. A.: Scalable Graph Exploration on Multicore Processors, SC '10, pp. 1–11 (2010).

[37] Frasca, M., Madduri, K. and Raghavan, P.: NUMA-aware Graph Mining Techniques for Performance and Energy Efficiency, SC '12, pp. 95:1–95:11 (2012).

[38] Tsujita, Y., Endo, T. and Fujisawa, K.: The Scalable Petascale Data-driven Approach for the Cholesky Factorization with Multiple GPUs, ESPM '15, pp. 38–45 (2015).