数理解析研究所講究録
第2005巻 2016年 32-42

エクサフロップス時代に向けた線形計算アルゴリズムの課題と研究動向

山本有作
電気通信大学 情報理工学研究科 情報・通信工学専攻／ JST-CREST
Yusaku Yamamoto
Graduate School of Informatics and Engineering, The University of Electro-Communications
/ JST-CREST

1 はじめに

スーパーコンピュータの性能は10年で1,000倍のペースで向上しており，この傾向が続くと，2018年頃にはエクサフロップスマシン，すなわち1秒間に10^{18}回の浮動小数点演算を実行可能なコンピュータが登場すると予想されている．エクサフロップスマシンでは，現在のスーパーコンピュータと比べてコア数がさらに増え，10^9個レベルになると見込まれる．また，メモリ階層がより深くなるとともに，演算性能とデータ転送性能との乖離がますます大きくなると予想される．さらに，部品点数の増加に伴い，故障確率が増大する可能性がある．このようなマシンを使いこなし，その性能を引き出すには，数値計算アルゴリズムの側でも大きな変革が必要となる．そこで，本報告では，数値計算の中でも特に線形計算に焦点を当てて，エクサフロップス時代における課題を明らかにし，その解決に向けて現在行われている研究のいくつかを取り上げて紹介する．

本報告の構成は以下の通りである．まず第2章では，現時点で予想されているエクサフロップスマシンのハードウェア特性を，「HPCI技術ロードマップ白書」[1]に基づき概観する．第3章では，大規模並列アプリケーション側の視点から，エクサフロップス時代における線形計算への要求を考える．第4章では，前の2つの章での考察を踏まえ，エクサフロップス時代における線形計算アルゴリズムの課題を整理する．第5章では，これらの課題の解決に向けて，現在行われている研究のいくつかを紹介する．最後に第6章でまとめを述べる．

2 エクサフロップスマシンのハードウェア特性

2.1 2018年のスーパーコンピュータ

2012年3月に発表された「HPCI技術ロードマップ白書」では，2018年に実現が予想されるスーパーコンピュータを，汎用型（従来型），容量・帯域重視型，演算重視型，メモリ容量削減型の4つのタイプに分類している．ここで，汎用型とは，現在のスーパーコンピュータのメモリ容量・帯域・演算性能をバランス良く向上させたタイプであり，「京」のように汎用的に様々な問題に適用可能である．容量・帯域重視型は，汎用型から演算性能を落として，メモリ性能により多くの資源を割くタイプである．演算重視型は，逆に，メモリ性能を落とし，演算性能により多くの資源を割くタイプである．メモリ容量削減型は，メモリ容量を極限まで削減し，オンチップメモリですべての計算を完結させるタイプである．条件として，「京」と同程度の消費電力（20MW），同程度の設置面積（2,000〜3,000m^2）を仮定した場合，各タイプで実現可能な総演算性能は，それぞれ200〜400PFLOPS，50〜100PFLOPS，1,000〜2,000PFLOPS，500〜1,000PFLOPSとされている（[1], 表2-5）．したがって，エクサフロップスに最も近いのは，演算重視型とメモリ容量削減型ということになる．しかし，総メモリ容量は，演算重視型が5〜10PByteに対し，メモリ容量削減型は0.1〜0.2PByteと極端に少なく，実行可能なアプリケーションが大きく制約される．以上より，現時点において，エクサフロップスマシンの最有力候補は演算重視型であると考えられる．

2.2 演算重視型スーパーコンピュータのハードウェア特性

そこで，以下では，演算重視型のアーキテクチャを前提として，エクサフロップスマシンのハードウェアの特徴と留意すべき点を概観する．

10^9 レベルの並列性 現在，CPU コアの動作周波数は GHz オーダーであるが，消費電力の問題から，この値は今後数年は大きく変わらない見込みである．そのため，エクサフロップスを実現するには，10^9 レベルの並列性が必要になる．この並列性はフラットではなく，演算器レベル，コアレベル，チップレベル，ノードレベルなど，多くの階層での並列性からなる．

複雑なメモリ階層 メモリについては，現在でもキャッシュメモリ，ノード内メモリ，他ノードのメモリなどの階層があるが，上で述べた多階層の並列性に対応して，メモリもより多層化，複雑化すると考えられる．

データ移動コストの増大 これまでのスーパーコンピューターの性能トレンドでは，演算性能の向上に，メモリアクセス性能や通信性能などのデータ転送性能の向上が追いついておらず，両者の乖離は今後も増大し続けると予想される．データ移動コストを，スループットとレイテンシに分けて考える．まずスループットであるが，前節で引用した予測によれば，総演算性能 1,000〜2,000PFLOPS の演算重視型マシンの総メモリ帯域は 5〜10PByte/s とされている．したがって，データ転送性能と演算性能の比は 0.005Byte/Flop となる．これは，倍精度実数データ（8Byte）を 1 個メモリから取ってきたら，1600 回の演算を行わないと，演算性能をフルに発揮できないことを意味する．一方,「京」では，総演算性能 10PFLOPS に対して総メモリ帯域は 5PByte/s なので，0.5Byte/Flop である．すなわち，ここで想定する演算重視型マシンでは，「京」に比べて実に 100 倍も主メモリアクセスの相対的コストが高いことになる．一方，レイテンシについては，コア間同期・通信レイテンシが 100ns （$\simeq$100 サイクル），ノード間通信レイテンシが 80〜200ns と，現在に比べてほとんど性能向上がないことが予想されている（[1], 表 2-3）．したがって，演算性能の向上と，ノード数増加によりレイテンシが生じる段数が増加することを考えると，実行時間に対するレイテンシの影響は，現在より遥かに大きくなる．これは，ベクトルの内積のような AllReduce 型の通信で特に顕著であり，たとえば 10^5 個のノード間で 2 進木により AllReduce を行う場合，数千サイクルの時間を要することになる．

電力消費の問題 エクサフロップスマシンでは，節電と発熱抑制の両面から，電力消費の抑制が重要である．その際，オフチップのデータ転送が大きな電力を消費することに留意すべきである．実際，前記白書の予測によれば，倍精度演算の消費電力は 1.1pJ/FLOPS（[1], 表 2-1）であるのに対し，オンチップのデータ転送は 2pJ，オフチップのデータ転送は 200pJ を消費する（[1], 2.1.4 項）．したがって，できる限りオフチップのデータ転送を抑えることは，性能面だけでなく，消費電力の面からも重要である．

部品数増加に伴う故障確率上昇 エクサフロップスマシンでは，LSI の微細化がさらに進むとともに，システムを構成する部品数が現在に比べて飛躍的に多くなる．そのため，α 線などによるソフトエラーの確率が上昇するとともに，部品の故障による障害も起きやすくなると考えられる．

3 大規模並列アプリケーションにおける要求の変化

以上では，今後の線形計算アルゴリズムの設計に大きな影響を及ぼす可能性があるエクサフロップスマシンの特性について論じた．一方，エクサフロップスの時代になると，アプリケーションプログラムからの線形計算に対する要求も，これまでとは変わってくる可能性がある．

弱スケーリングから強スケーリングへ　これまでの科学技術計算は，演算能力やメモリが増大するにつれ，ますます大きな問題を解くという方向で発展してきた．そのため，大規模並列計算においては，コア数に比例して問題規模も大きくする弱スケーリングの条件下で性能が出ればよいとされてきた傾向がある．しかし，現在のスーパーコンピュータ，たとえば「京」では，解ける問題のサイズはかなり大きくなっており，必ずしも全ての応用で問題サイズをより大きくしたいという状況ではなくなっている．たとえば，分子動力学のある応用では，問題サイズは固定して，代わりに，超長時間，たとえば 100 万ステップにわたる時間発展を追いたいという需要がある．この場合，時間方向の並列化は困難であるから，実用的な時間で計算を終えるには，1 ステップの計算時間をできるだけ短縮する必要がある．すなわち，問題サイズを一定にしてコア数を増やすという強スケーリングの下での並列性能が重要となる．

計算量のオーダーの低減への要求　多くの線形計算では，計算量は問題サイズに対して線形より速く増加する．特に密行列計算では，LU 分解，QR 分解，特異値分解など，行列サイズの 3 乗，すなわちデータ量の 1.5 乗で計算量が増える場合が多い．この場合，データ量に比例してコア数を増やしても，計算時間は行列サイズとともに増大し，超大規模問題では現実的な時間で計算が終わらなくなる可能性がある．そのため，何らかの意味で近似を導入するなどして，計算量のオーダーを減らしたアルゴリズムが求められる．

高精度演算利用の必要性　これは，アプリケーション側からの要求というよりも，線形計算自体の問題であるが，超大規模行列，たとえば密行列で 100 万元を超えるような行列を扱う場合，倍精度では演算精度が不足し，十分な有効桁数の解が得られなくなる．そのため，4 倍精度などの高精度演算を利用するか，あるいは精度面からアルゴリズムを見直すことが必要となる．

4　線形計算アルゴリズムの課題

以上の考察を踏まえると，エクサフロップス時代に向けた線形計算アルゴリズムの研究では，次のような課題に取り組む必要があると考えられる．

(1) 10^9 レベルの並列性と階層性への対処　エクサフロップスマシンの持つ演算能力を活用するために，10^9 レベルの並列性が必要である．また，ハードウェアの持つ階層的な並列性に対応して，アルゴリズムも階層的な並列性を持つことが望ましい．

(2) データ移動量の削減　エクサフロップスマシンでは，主メモリアクセスやノード間通信など，データ移動のコストが非常に大きい．そこで，データ移動量をできるだけ小さくする必要がある．そのためには，アルゴリズムのデータ再利用性を向上させ，データがキャッシュなど上位のメモリにある間に，できるだけ集中して演算を行うことが必要である．これは，性能向上のためだけでなく，消費電力の削減にも役立つ．

(3) データ移動回数の削減　データの移動量によらずに移動 1 回ごとにかかる固定コストを削減することも重要である．たとえば，ノード間通信 1 回ごとにかかるコスト，コア間同期 1 回ごとにかかるコスト，主メモリアクセスにおいて，アクセス対象のアドレスに不連続が生じた場合にかかるコストなどが，この例である．これらのコストを削減するには，アルゴリズムの計算粒度を大きくし，同期やデータ移動をできるだけまとめて行うとともに，メモリアクセスをできるだけ連続化することが重要である．

(4) データの読み出しエラーや通信エラーがあっても計算を続けられる耐故障性　ハードウェアのエラーに対しては，基本的には，ハードウェアや OS のレベルで対処してもらうか，あるいはチェックポインティングのような汎用的な手法で対処することになると考えられる．しかし，もしアルゴリズム自体に冗長性を導入し，演算における結果不正や，通信においてデータが到着しないなどの障害があっても計算が破綻せずに進行するようにできれば，大きなメリットがあると思われる．

(5) ある程度の誤差を許容することで計算量のオーダーを引き下げられるアルゴリズム　超大規模問題を実用的な時間で解くには，ある程度の（確率的）誤差を許容することで，計算量のオーダーを引き下げるという方向の研究も重要である．たとえば，テンソルの低ランク近似に基づくアルゴリズムや，確率的アルゴリズムなどである．特に，最近話題となっているビッグデータの解析では，元々の行列がある母集団からのサンプリングにより得られた標本誤差を含むデータであるから，たとえば特異値分解などの処理を行う場合，従来のように丸め誤差レベルまでの精度を求める必要はないように思われる．このような応用は，確率的アルゴリズムと親和性が高いと考えられる．

(6) 強スケーリングの意味で効率的なアルゴリズム　弱スケーリングに重点を置いたこれまでの並列アルゴリズムに対して，今後は，強スケーリングの意味で効率的なアルゴリズムも重要になる．強スケーリングの条件下では，通信・同期時間やメモリアクセスのレイテンシが支配的になることから，これらのコストを削減できるアルゴリズムの開発が必要になる．大粒度のアルゴリズムは，この観点からも有効である．

(7) 高精度演算を有効に活用できるアルゴリズム　エクサフロップス時代の大規模計算では，倍精度演算では精度が不足する場面が今まで以上に多くなると考えられる．しかし，演算のすべてを 4 倍長などの高精度演算にすると，計算時間とメモリの両面でコストが大きい．そこで，誤差解析あるいは経験に基づき，必要な部分のみを高精度演算で行う混合精度型の演算が重要となる．また，入力データの内容に基づいて演算順序を変更するなどの工夫により，必ずしも高精度演算を使わずに丸め誤差の影響をできるだけ抑制する技法も重要になると思われる．

5　線形計算アルゴリズムの研究動向

本章では，前章で述べた課題の解決に向けて，線形計算アルゴリズムの分野で近年行われている研究のいくつかを紹介する．

5.1　10^9 個のコアを活用できる並列性

まず，エクサフロップスマシンにおいて必須となる 10^9 レベルの並列性について，密行列と疎行列に分けて述べる．

5.1.1　密行列の場合

正方行列に対するアルゴリズム　密行列に対して広く使われている線形計算アルゴリズムとして，LU 分解，QR 分解，対称固有値問題のための 3 重対角化，非対称固有値問題のための Hessenberg 化などがある．これらは行列の行または列を 1 列（行）ずつ消去してゆくという構造を持ち，$n \times n$ の正方行列に適用する場合，全体の計算量は $O(n^3)$，各ステップでの計算量は $O(n^2)$ である．したがって，たとえば $n = 10^6$ 程度であれば，10^9 コアでも 1 ステップ・1 コアあたり 10^3 回程度の演算を担当でき，並列性としては十分である．ただし，これらのアルゴリズムでは，消去のためのピボット行・列の生成・通信というステップが存在し，そこが性能ネックとなりうる．そのため，これらを消去演算とオーバーラップさせ，実行時間を隠蔽するためのスケジューリング技術が重要となる．最近では，ピボット生成や消去などの演算をブロック単位でタスクとして定義し，それらの依存関係を無閉路有向グラフ（Directed Acyclic Graph）で表現して，汎用的な手法を用いてスケジューリングを行う DAG スケジューリングという技法が開発されている [2]．これを実現する DAGuE というソフトウェアも公開されており，密行列の LU 分解，QR 分解などで高い並列性能が確認されている．

帯行列・縦長行列に対するアルゴリズム　一方，帯行列や縦長行列に対するアルゴリズムも重要である．具体的には，帯行列の LU 分解，帯行列の 3 重対角化，縦長行列の QR 分解，ベクトル逐次添加型の直交化などがある．これらは，行列の縦方向のサイズを n，帯幅または横方向のサイズを b とするとき，$n \times b$ のデータに対する演算であり，全体の計算量は $O(n^2b)$ または $O(nb^2)$ となる．また，従来から使われているアルゴリズムの並列性は $O(b^2)$（帯行列の LU 分解，3 重対角化），$O(nb)$（縦長行列の QR 分解），あるいは $O(b)$（ベクトル逐次添加型の直交化）と小さい．そのため，超並列環境向けには，並列性を向上できる新たなアルゴリズムが必要となる．これらの例としては，帯行列に対する分割統治型の LU 分解法 [3][4][5]，Compact WY 表現に基づくベクトル逐次添加型の直交化アルゴリズム [6][7] などがある．

5.1.2　疎行列の場合

疎行列に対しては，連立 1 次方程式の求解，固有値計算，行列関数の計算などにおいて，部分空間への射影に基づくアルゴリズムが広く使われている．たとえば，連立 1 次方程式の求解のための Krylov 部分空間法全般，固有値計算のための Lanczos 法，Arnoldi 法，Jacobi-Davidson 法などがその例である．これらの解法では，ステップごとに部分空間を拡大してゆくという操作を行う．しかし，この操作は本質的に逐次的であるため，並列化は 1 ステップの内部に限られる．1 ステップの演算は，通常，行列とベクトルの積が大部分を占めるため，並列性は $O(nz)$（nz は行列の非ゼロ要素数）となる．

そこで，並列性を向上させるため，いくつかの手法が提案されている．その一つは，Krylov 部分空間法において複数の右辺ベクトルあるいは初期ベクトルを用いるブロック Krylov 部分空間法である．これについては，次節で述べる．もう一つは，求める部分空間のみを抽出するフィルタを数値的に構成し，初期値として与えたベクトル（群）から，必要な部分空間を一気に抽出する手法である．例としては，固有値問題のための櫻井・杉浦法 [8] や，filter-diagonalization 法 [9] が挙げられる．これらの手法では，フィルタの作用を，入力ベクトルを右辺とする複数の連立 1 次方程式を解くことで実現するが，これらの方程式は独立に解けるため，その部分で新たな並列性が生まれる．さらに，スペクトルの複数の部分を求めたい場合は，複数のフィルタを適用することになり，そこでさらに並列性が生まれる．このため，これらの解法は，大粒度の超並列性を持つ新しい解法として，今後が期待されている．

5.2　データ移動の削減

次に，データ移動量と移動回数を削減するための工夫について，密行列と疎行列の場合に分けて述べる．

5.2.1　密行列の場合

いま，設定として，コアが 1 個，キャッシュが 1 階層の計算機で，キャッシュと主メモリとの間のデータ移動を最小化したい場合を考える．実際のマシンでは，キャッシュは多階層で，他のノードとの通信も考慮する必要があるが，その場合にも，以下に述べる手法は拡張できる．なお，キャッシュの容量を M ワード，行列サイズを n とする．

密行列に対する線形計算では，古くからブロック化（タイル）アルゴリズムが使われてきた．このアルゴリズムでは，行列をサイズ $\sqrt{M/3} \times \sqrt{M/3}$ のブロックに分割する．これは，ブロック 3 個がちょうどキャッシュに入るサイズである．その上で，各ブロックを行列要素のように見て，計算を行う．これにより，行列乗算，LU 分解，コレスキー分解，QR 分解，直交変換による帯行列化，直交変換による帯 Hessenberg 化など，様々なアルゴリズムがブロック化できる．ブロック化アルゴリズムでは，演算の主要部分がブロックどうしの乗算となり，これがキャッシュ上で実行可能のため，キャッシュ利用効率がよい．ブロック化アルゴリズムは，LAPACK や ScaLAPACK をはじめとする線形計算ライブラリで広く使われている．

最近の研究として，様々なブロック化アルゴリズムについて，上記の設定の下でデータ移動量の意味での最適性が証明されている．たとえば，ブロック化コレスキー分解については次の定理が示されている [10].

定理（Ballard, Demmel, Holtz and Schwartz, 2009） ブロックサイズを $\sqrt{M/3}$ としたブロック化コレスキー分解は，上記の設定の下で，キャッシュと主メモリの間のデータ移動量をオーダーの意味で最小化する．

この定理の証明では，行列乗算についてはデータ移動量の下界がわかっていることに着目する．そして，コレスキー分解を用いて行列乗算を計算するアルゴリズムを構築する．これにより，定数倍の差を除いて，コレスキー分解におけるデータ移動量の下界が行列乗算におけるデータ移動量の下界以上であることが言える．さらに，ブロック化コレスキー分解におけるデータ移動量が，行列乗算におけるデータ移動量の下界を達成することを示すことで，定理が証明される．

以上では，データ移動量について論じたが，データ移動回数についても検討する必要がある．いま，1回のデータ移動では，主メモリの連続した領域しかキャッシュに持ってこられないと仮定する．このとき，通常の行列格納形式では，ブロック化コレスキー分解は移動回数最小にならない．なぜなら，ブロックを1個取り出した場合，その内部のアドレスは一般には連続でないからである．そこで，ブロックの中が連続アドレスになるように，格納形式の変更を行う．これにより，データ移動回数も最小となることが示される．データ移動量とデータ移動回数の両方が最小になるアルゴリズムを，communication-optimal なアルゴリズムと呼ぶ．

これまでに，データ移動量・回数の下界が知られている線形計算としては，行列乗算（$O(n^3)$ のアルゴリズムと Strassen のアルゴリズムの両方），LU 分解，コレスキー分解，QR 分解，最小2乗法，固有値分解，特異値分解などがある．ただし，下界がわかっているとは言っても，これらすべての計算について，下界を達成するアルゴリズムが知られているわけではない．両方の下界を達成する communication-optimal なアルゴリズムの開発は，現在，活発な研究テーマとなっている．

5.2.2 疎行列の場合

疎行列に対するアルゴリズムの例として，Krylov 部分空間法を取り上げる．Krylov 部分空間法は，Krylov 部分空間 $K_m(A;b) = \mathrm{span}\{\mathbf{b}, A\mathbf{b}, A^2\mathbf{b}, \ldots, A^{m-1}\mathbf{b}\}$ の中で近似解を求めていく解法であり，連立1次方程式の求解，固有値計算，行列関数の計算など，幅広い応用を持つ．演算の主要部分は疎行列ベクトル積 $\mathbf{y} = A\mathbf{x}$ であり，また，1ステップ中に複数回の内積とノルム計算が存在する．データ移動の観点からの問題点としては，(i) 行列ベクトル積 $\mathbf{y} = A\mathbf{x}$ において，行列 A の各要素は1回しか計算に利用されず，データの再利用性が低いこと，(ii) 内積やノルム計算は全ノードでの AllReduce を必要とし，レイテンシの影響が大きいことが挙げられる．そこで，行列ベクトル積におけるデータ再利用性の向上と，複数の内積をまとめて AllReduce の回数を削減することが課題となる．以下では，GMRES 法を例として，そのための手法を紹介する．

GMRES 法は，Krylov 部分空間に基づく最も素直な連立1次方程式の解法である．右辺ベクトル $\mathbf{b}$ から出発して，A をかける操作と直交化を繰り返し，部分空間を拡大しながら正規直交基底 $\mathbf{q}_1, \mathbf{q}_2, \mathbf{q}_3, \ldots$ を生成し，各ステップにおいて，残差 $\|\mathbf{r}_m\|_2 = \|\mathbf{b} - A\mathbf{x}_m\|_2$ が最小になるよう近似解 $\mathbf{x}_m$ を更新してゆく．計算量は，正規直交基底を生成する部分，すなわち行列ベクトル積と，ベクトルの逐次添加型の直交化が支配的である．以下では，この部分を対象とした手法を2つ紹介する．

ブロック GMRES 法 [11] 複数（b 本）の初期ベクトル $R^{(0)} = [\mathbf{r}_1^{(0)}, \ldots, \mathbf{r}_b^{(0)}]$ から出発し，ブロック Krylov 部分空間 $K_m(A; R^{(0)})$ 内で解を求める手法である．各ステップでの行列ベクトル積 $\mathbf{y} = A\mathbf{x}$ が行列乗算（$Y = AX$）に置き換わるため，行列 A のアクセス回数は同じで演算量が b 倍となり，A の再利用性が b 倍向上する．また，1ステップあたりの AllReduce の回数は（データ量は増えるが）不変である．この方法を使って右辺ベクトルのみが異なる b 組の方程式を解く場合，1ステップあたりの行列アクセス回数，AllReduce の回数は普通の GMRES 法と同じため（表1参照），実行時間の増加は b 倍以下となる．さらに，各方程式の近似解を $m \times b$ 次元の広い空間 $K_m(A; R^{(0)})$ 内で求めているため，収束性向上による加速

の効果も得られる．したがって，ブロックGMRES法，より一般にはブロックKrylov部分空間法は，このような場合には極めて有効な解法となる．

Table 1: GMRES法とブロックGMRES法との比較（1ステップあたり）

	GMRES法	ブロックGMRES法
演算量（行列ベクトル積）	1 ($\mathbf{y} = A\mathbf{x}$)	b ($Y = AX$)
行列 A のアクセス回数	1 ($\mathbf{y} = A\mathbf{x}$)	1 ($Y = AX$)
AllReduceの回数	1	1

k-step GMRES法 [12]　GMRES法において，行列ベクトル積 $A\mathbf{r}^{(m)}, A^2\mathbf{r}^{(m)}, \ldots, A^k\mathbf{r}^{(m)}$ を一度に行ってKrylov部分空間を一度に k 次元拡大し，その後に正規直交基底を生成する手法である．A が有限要素法や差分法などから生じる行列の場合，A の作用はメッシュ上で局所的であるため，メッシュ上の部分領域に相当する $\mathbf{r}^{(m)}$ の要素をキャッシュ上に取ってくれば，それらのみを使って $A\mathbf{r}^{(m)}, A^2\mathbf{r}^{(m)}, \ldots, A^k\mathbf{r}^{(m)}$ の一部を計算できる．これにより，A，$\mathbf{r}^{(m)}$ の再利用性が向上する．また，直交化も k 本分をまとめて行うことができ，AllReduceの回数を $1/k$ に削減できる（表2参照）．

この方法の欠点は，直交化前の基底 $A\mathbf{r}^{(m)}, A^2\mathbf{r}^{(m)}, \ldots, A^k\mathbf{r}^{(m)}$ が線形従属に近い場合に，収束性が悪化することである．そこで，線形独立性を高めるため，A の単項式の代わりに適当な直交多項式 $\{p_i(x)\}_{i=0}^{k}$ を使い，3項間漸化式を用いて，基底 $p_1(A)\mathbf{r}^{(m)}, p_2(A)\mathbf{r}^{(m)}, \ldots, p_k(A)\mathbf{r}^{(m)}$ を計算するなどの工夫が検討されている [12][13].

Table 2: GMRES法と k-step GMRES法との比較（1ステップあたり）

	GMRES法	k-step GMRES法
演算量（行列ベクトル積）	1	$1+\alpha$
行列 A のアクセス回数	1	1/k
AllReduceの回数	1	1/k

5.3　アルゴリズムレベルでの耐故障性

本節では，アルゴリズムレベルでの耐故障性を持つ線形計算アルゴリズムについて述べる．想定する状況として，複数のノードが通信を行いつつ協調して計算し，そのうち1個のノードが不正な結果を返すか，あるいは結果を返さない（通信のタイムアウト）場合を考える．このような状況においても計算が破綻せずに進行するアルゴリズムを作りたい．ただし，その場合，精度の劣化あるいは収束性劣化は許容するとする．また，計算の任意の箇所でエラーが起きうるとすると，アルゴリズムの設計が非常に難しくなるため，計算の一部は高信頼モードあるいは高信頼ハードウェア（エラーは起こらないが計算コスト大）で行ってもよいとする．

まず，予備的な考察として，複数のノードの結果を集めて部分空間を改良するタイプのアルゴリズムは，耐故障性と親和性が高いことがわかる．また，大粒度並列性は，あるノードで起こったエラーの結果が全体を汚染する頻度を減らすため，耐故障性にとっても有利であることがわかる．そこで，以下では，このようなタイプのアルゴリズムを3つ紹介する．

MERAM（Multiple Explicitly Restarted Arnoldi Method）[14]　Arnoldi法に基づく固有値計算のための耐故障性アルゴリズムである．アイディアは非常に単純であり，以下の処理を繰り返して計算を

行う．

(1) P 個のノードで，異なる初期ベクトルを用いて独立に Arnoldi 法を実行する．

(2) 各ノードで作った Krylov 部分空間を合わせて，大きな部分空間を生成する．

(3) その中で，P 本の初期ベクトルを新たに生成し，リスタート用のベクトルとして各ノードに分配する．

このうち，(1) を低信頼モード，(2)，(3) を高信頼モードで実行する．計算の大部分は (1) のステップで行われるため，高信頼モードでの計算量は小さい．また，(1) において，結果を返さないノードがあっても，和空間の次元が小さくなって収束性が落ちるだけで，計算は破綻しない．結果不正によって，あるノードが全くおかしな部分空間を返したとしても，その部分空間が加わることで，収束性が悪化することはない．このようにして，MERAM はある条件の下での耐故障性を実現していることがわかる．

Fault-Tolerant GMRES 法 [15]　Flexible GMRES 法の枠組みを利用した，連立 1 次方程式求解のための耐故障性アルゴリズムである．Flexible GMRES 法では，ステップごとに異なる前処理を用いることが許される．そこで，このアルゴリズムでは，結果不正を，不適切な前処理と解釈して計算を続行する．その結果，そのステップでは無駄に部分空間が大きくなるが，精度に悪影響が生じることはない．

櫻井・杉浦法 [16]　固有値計算のための櫻井・杉浦法では，フィルタを用いて，入力のベクトル群から求めたい部分空間を抽出する．フィルタとしては，複素平面上で求めたい固有値を囲む閉曲線上でのレゾルベントの周回積分を用い，これを数値積分で近似して計算する．いま，故障によって 1 個の積分点での結果が求まらなかった場合，結果が求まった積分点のみを用いる積分公式を新たに構築して用いれば，精度は落ちるものの，計算は続行できる．櫻井・杉浦法では，このようにして耐故障性を実現できる．

以上では，疎行列向けの耐故障性アルゴリズムについて述べた．一方，密行列向けの解法では，連立 1 次方程式の求解でも固有値計算でも直接解法が基本であり，部分空間法のような冗長性がないため，1 箇所の計算間違いで結果に壊滅的な影響が生じる．そのため，アルゴリズムレベルでの耐故障性の実現は原理的に難しく，チェックポインティングなどの汎用的な技法に頼るのが現実的ではないかと考えられる．ただし，計算過程が可逆で，かつ，逆向きの計算が安定に行える場合には，エラーに気付いた時点で元に戻って計算をやり直すことも可能である．QR 分解など，直交行列による変換を行うアルゴリズムはその例であり，実際，CPU-GPU ハイブリッド環境における耐故障性 QR 分解アルゴリズムが提案されている [17].

5.4　確率的アルゴリズムによる計算量のオーダー低減

ある程度の（確率的）誤差を許容することで，線形計算の計算量のオーダーを引き下げるという研究の例として，本節では確率的アルゴリズムによる CX 分解の計算を紹介する．

行列の特異値分解は，行列の最良の低ランク近似を与える分解として知られている．この性質に基づき，特異値分解は，画像処理，信号処理，情報検索などの分野で幅広く利用されている．しかし，$m \times n$ 行列 A（$m \geq n$）に対し，特異値分解の計算量は $O(mn^2)$ と大きい．そのため，いわゆるビッグデータを扱う分野では，特異値分解をそのまま用いることが困難になってきている．

そこで，特異値分解の代替手法として CX 分解が注目されている．CX 分解とは，C を A の列ベクトルを k 本（$1 \leq k \leq n$）選んでできる $m \times k$ 行列，X を $k \times n$ 行列とするとき，$\|A - CX\|_{\mathrm{F}}$（$\|\cdot\|_{\mathrm{F}}$ はフロベニウスノルム）をできるだけ小さくするような C と X を求める分解である．これは，A の特徴を最もよく表す k 本の列ベクトルを選ぶことに相当する．C を決めれば，X は $X = C^{+}A$（C^{+} は C の Moore-Penrose 逆行列）と選ぶのが最適であることが示されるため，問題は C を選ぶことに帰着する．CX 分解では，C の列が元の A の列そのものであるため，代表データとして解釈しやすいという利点がある．これに対して特異値分解では，たとえば A の要素が整数でも，特異ベクトルの要素は一般に整数にならないため，解釈

が難しい場合がある．このような利点と，以下で述べるような高速な確率的アルゴリズムが存在するという利点から，CX 分解の利用は急速に広まっている．

C の列の選択には，statistical leverage と呼ばれる量を用いる [18]．ここで，A のランク k の打ち切り型特異値分解を $A_k = U_k \Sigma_k V_k^\top$ とするとき，$V_k^\top$ の第 j 列の statistical leverage p_j は次のように定義される．

$$p_j = \frac{\| \left(V_k^\top\right)^{(j)} \|_2}{k} \tag{1}$$

ただし，$\left(V_k^\top\right)^{(j)}$ は $V_k^\top$ の第 j 列ベクトルである．このとき，確率 p_j に従って A の列をサンプリングすると，確率 0.9 以上で次の誤差評価が成り立つことが示せる．

$$\|A - CC^+A\|_{\mathrm{F}} \leq (1+\epsilon)\|A - A_k\|_{\mathrm{F}}. \tag{2}$$

ただし，ϵ はサンプリング回数に依存する微小量である．上式は，こうして得られた CX 分解により，高い確率で，A_k を高い相対精度で近似できることを意味する．ただし，以上の方法は，CX 分解の計算法としては実用的でない．CX 分解を計算するために，打ち切り型特異値分解の V_k を使っているからである．

そこで，statistical leverage に対する高速な近似計算アルゴリズムが開発されている [19]．この方法のアイディアは，A に左から適当な直交行列をかけることで，一様な statistical leverage を持つ行列 A' に変換することである．そのために，Johnson & Lindenstrauss の補題と高速 Walsh-Hadamard 変換を使う．A' に対しては，列の一様なサンプリングを行えば，上記の確率的誤差評価を持つ CX 分解が求められる．その上で，A' の CX 分解から A の CX 分解を求めればよい．この方法により，相対誤差の意味での確率的誤差評価を持つ CX 分解の計算が $O(mn \log m)$ の計算量で可能になり，大きなブレークスルーとなった．

5.5 強スケーリングの意味で効率的なアルゴリズム

本章の最後では，強スケーリングの意味で効率的なアルゴリズムを目指す研究について，固有値計算を例として紹介する．

実対称行列の全固有値・固有ベクトルを求める問題は，分子軌道法をはじめ，科学技術計算の様々な分野で重要である．このタイプの計算では，「京」をはじめとするペタスケールの計算機を使って百万元規模の問題も解かれているが，一方で，1 万元程度の中規模問題に対する需要も多い．そこで，行列サイズを $n = 10,000$ に固定し，ノードを何個使ってもよいから，できるだけ高速に解きたいという問題設定を考える．このような状況は，たとえば分子軌道法で生じうる．分子軌道法において，行列を生成する多電子積分は，計算量が $O(n^4)$ 以上と多く，かつ並列性も高い．そのため，1 万ノード以上を用いて多電子積分を並列化する例もある．その場合，演算量が $O(n^3)$ である固有値計算の部分が目立ってくるため，今度はその並列化が課題となる．ところが，分子軌道法における典型的な行列サイズである $n = 10,000$ の場合，標準的な線形計算ライブラリである ScaLAPACK では，高々数百ノード程度で性能が飽和する．そのため，プログラムの実行用に確保した 1 万ノードの大部分は，固有値計算部分では遊んでしまうことになる．このような状況下では，たとえ並列化効率が低くても，確保したノードを最大限に使うことで実行時間を短縮できれば望ましい．

ScaLAPACK で性能が飽和する理由は，固有値計算の前処理である 3 重対角化において，小粒度の通信が多発するからである．実際，「京」上での $n = 10,000$ の実行結果を解析すると，実行時間の 70%以上を通信が占めており，さらに，その大部分を通信のスタートアップ時間，すなわち，通信量ではなく通信回数に比例してかかる時間が占めている．そこで我々は，中規模問題を 1 万ノード規模で実行するのに適したアルゴリズムとして，ヤコビ法の一種であるブロックヤコビ法に着目した．ブロックヤコビ法は，3 重対角化に基づくアルゴリズムと比べ，計算量は 10 倍程度と多いが，通信回数は，3 重対角化の $O(n \log_2 P)$ に対して $O(\sqrt{P} \log_2 P * Iter)$（$P$：ノード数，Iter：反復回数，$P$ ノードでのブロードキャストは通信回数 $\log_2 P$ 回と換算）となる．$n = P = 10,000$ のとき，後者の回数は，前者と比べてずっと少ない．そのため，通信

オーバーヘッドが支配的となる状況では，ブロックヤコビ法が 3 重対角化に基づくアルゴリズムより高速となる可能性がある．

そこで，ブロックヤコビ法のアルゴリズムを「京」上で実装し，行列サイズを $n = 10,000$ に固定して ScaLAPACK と性能を比較したところ，スケーラビリティの面ではブロックヤコビ法が優位であり，その実行時間は P を増やすにつれて単調に減少し，$P = 10,000$ のときは ScaLAPACK の最速値を上回ることがわかった [20]．ブロックヤコビ法に関しては，ブロックの消去順序の最適化，前処理などにより，さらに高速化できる可能性があり，強スケーリングの条件下では優位な解法になりうると考えられる．

以上からわかるように，強スケーリングの条件下でのアルゴリズム設計においては，通信・同期オーバーヘッドの削減が最も重要であり，そのためならば，演算量をある程度増やしてもよい．このような考え方に基づき，最近，中務らは，極分解を用いた通信削減型の新しい固有値・特異値計算アルゴリズムを提唱しており [21]，今後の展開が期待される．

6 終わりに

今後登場が予想されるエクサフロップスマシンでは，多階層の超並列性，データ移動コストの増大，故障確率の上昇などが課題となる．また，応用分野からの要請としては，強スケーリングでの並列化効率がより重視されるとともに，超大規模問題を現実的な時間で解くために計算量のオーダーの削減への要求が高まる．そのため，今後の線形計算アルゴリズムの研究では次の点が重要になると考えられる．

- 10^9 レベルの並列性と階層性への対処
- データ移動量・移動回数の削減
- アルゴリズムレベルでの耐故障性
- （確率的）近似による計算量のオーダーの削減
- 強スケーリングの意味での効率性

本報告では，これらの方向に沿った最近の研究をいくつか紹介した．本報告では主にアルゴリズムに焦点を当てて論じたが，複雑・高度化するアーキテクチャに対応する実装技術も大きなテーマであり，自動チューニング技術など，今後発展が期待される技術も多い．これらについては，機会があれば改めて論じたい．

謝辞

研究集会「応用数理と計算科学における理論と応用の融合」において発表の機会を下さった降旗大介先生，谷口隆晴先生に感謝いたします．また，日頃からご指導下さっている杉原正顯先生と，有益なコメントを下さった中務佑治さんに感謝いたします．本研究は科学研究費補助金および JST-CREST「ポストペタスケールに対応した階層モデルによる超並列固有値解析エンジンの開発」の補助を受けている．

References

[1] HPCI ロードマップ白書，http://open-supercomputer.org/wp-content/uploads/2012/03/hpci-roadmap.pdf, 2012 年 3 月.

[2] G. Bosilca, A. Bouteiller, A. Danalis, T. Herault, P. Lemarinier and J. Dongarra: DAGuE: A Generic Distributed DAG Engine for High Performance Computing, *Parallel Computing*, Vol. 38, No. 1-2, pp. 27–51 (2012).

[3] M. Hegland: Divide and Conquer for the Solution of Banded Linear Systems of Equations, in *Proceedings of the Fourth Euromicro Workshop on Parallel and Distributed Processing*, IEEE Computer Society Press, pp. 394–401 (1996).

[4] 山本有作, 猪貝光祥, 直野健: 非対称三重対角行列向けの並列連立一次方程式解法, 情報処理学会論文誌, Vol. 42, No. SIG9 (HPS), pp. 19–27 (2001).

[5] E. Polizzi and A. H. Sameh: A Parallel Hybrid Banded System Solver: The SPIKE Algorithm, *Parallel Computing*, Vol. 32, No. 2, pp. 177–194 (2006).

[6] H. Walker, Implementation of the GMRES Method Using Householder Transformations, *SIAM Journal on Scientific and Statistical Computing*, Vol. 9, No. 1, pp. 152–163 (1988).

[7] Y. Yamamoto and Y. Hirota: A Parallel Algorithm for Incremental Orthogonalization based on the Compact WY Representation, *JSIAM Letters*, Vol. 3, pp. 89–92 (2011).

[8] T. Sakurai and H. Tadano: CIRR: A Rayleigh-Ritz Type Method with Contour Integral for Generalized Eigenvalue Problems, *Hokkaido Mathematical Journal*, Vol. 36, No. 4 pp. 669–918 (2007).

[9] S. Toledo and E. Rabani: Very Large Electronic Structure Calculations Using an Out-of-Core Filter-Diagonalization Method, *Journal of Computational Physics*, Vol. 180, pp. 256–269 (2002).

[10] G. Ballard, J. Demmel, O. Holtz and O. Schwartz: Communication-Optimal Parallel and Sequential Cholesky Decomposition, *SIAM Journal on Scientific Computing*, Vol. 32, No. 6, pp. 3495-3523 (2010).

[11] V. Simoncini and E. Gallopoulos: Convergence Properties of Block GMRES and Matrix Polynomials, *Linear Algebra and Its Applications*, Vol. 247, No. 1, pp. 97–119 (1996).

[12] M. Hoemmen: Communication-Avoiding Krylov Subspace Methods, Ph. D Thesis, Computer Science Division, University of California, Berkeley, 2010.

[13] Z. Bai: A Newton Basis GMRES Implementation, *IMA Journal of Numerical Analysis*, Vol. 14, pp. 563–581 (1994).

[14] N. Emad, S. Petiton and G. Edjlali: Multiple Explicitly Restarted Arnoldi Method for Solving Large Eigenproblems, *SIAM Journal on Scientific Computing*, Vol. 27, No. 1, pp. 253–277 (2005).

[15] J. Elliott, M. Hoemmen and F. Mueller: Evaluating the Impact of SDC on the GMRES Iterative Solver, http://arxiv.org/abs/1311.6505.

[16] 白砂渓, 櫻井鉄也: 周回積分を用いた固有値解法の耐障害性について, 日本応用数理学会平成 23 年研究部会連合発表会, 2011 年 3 月.

[17] P. Du, P. Luszczek, S. Tomov and J. Dongarra: Soft Error Resilient QR Factorization for Hybrid System, *Journal of Computational Science*, Vol. 4, No. 6, pp. 457–464 (2013).

[18] C. Boutsidis, M. W. Mahoney and P. Drineas: An Improved Approximation Algorithm for the Column Subset Selection Problem, http://arxiv.org/abs/0812.4293.

[19] P. Drineas, M. Magdon-Ismail, M. W. Mahoney and D. P. Woodruff: Fast Approximation of Matrix Coherence and Statistical Leverage, *Journal of Machine Learning Research*, Vol. 13, pp. 3475–3506 (2012).

[20] 工藤 周平, 高橋 佑輔, 深谷 猛, 山本 有作: ブロックヤコビ法に基づく固有値解法の超並列計算機上での実装, 日本応用数理学会 2013 年度年会, アクロス福岡, 2013 年 9 月.

[21] Y. Nakatsukasa and N. J. Higham: Stable and Efficient Spectral Divide and Conquer Algorithms for the Symmetric Eigenvalue Decomposition and the SVD, *SIAM Journal on Scientific Computing*, Vol. 35, No. 3, pp. A1325-A1349 (2013).