

行列計算における高速アルゴリズム — ポストペタ時代に向けた線形計算アルゴリズムの 課題と研究動向 —

2019年6月27日

計算科学技術特論A

電気通信大学 情報理工学研究科 情報・ネットワーク工学専攻
山本 有作

本講義の構成

- 第1回(6月20日)

- 「大規模連立1次方程式の反復解法」

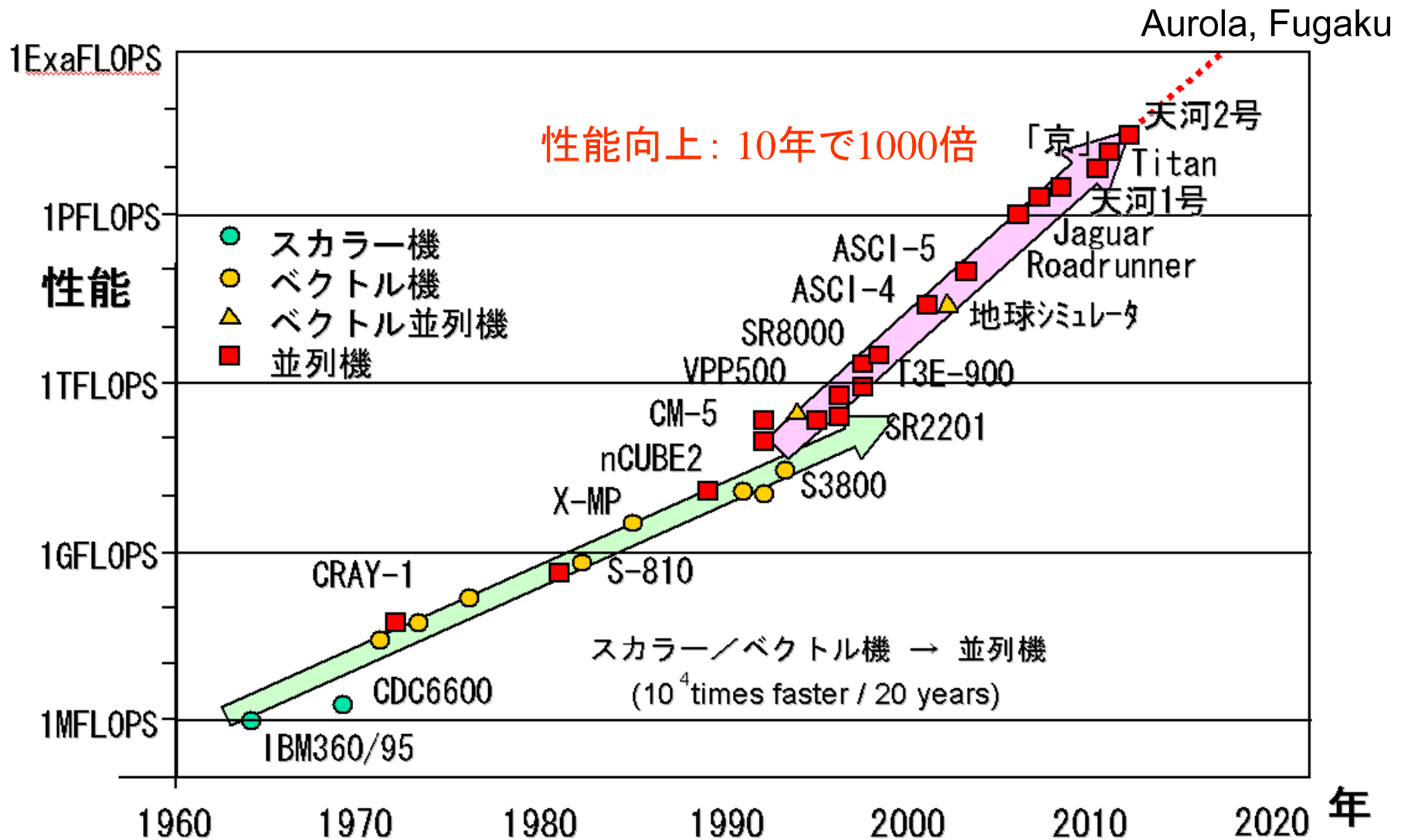
- 代表的な行列計算である連立1次方程式の解法について, 現在最も広く利用されているクリロフ部分空間法の基礎を紹介する

- 第2回(6月27日)

- 「ポストペタ時代に向けた線形計算アルゴリズムの課題と研究動向」

- 様々な行列計算について, ポストペタ計算機上での実装における課題と, その解決のための取り組みを紹介する

はじめに: スーパーコンピュータの性能動向



2021~22年にはエクサフロップスマシンが登場の見込み

本講義の目的

- 本講義の目的
 - エクサフロップス時代に向けた数値計算アルゴリズムの課題を、線形計算に焦点を当てて考える
 - 最近の研究動向について、簡単なサーベイを行う
- 目次
 - エクサフロップスマシンのハードウェア特性
 - 線形計算アルゴリズムの技術課題
 - 最近の研究動向

2018年のスーパーコンピュータ：4つのタイプ

- 汎用型 「京」の後継
 - メモリ容量・帯域・演算性能をバランス良く向上
 - 「京」のように汎用的に様々な問題に適用可能
- 容量・帯域重視型 ベクトル計算機タイプ
 - 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
- 演算重視型 メニーコアタイプ
 - メモリ性能を落とし、演算性能により多くの資源を割く
- メモリ容量削減型
 - 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

各タイプの性能諸元(予測値)

- 条件

- 「京」と同程度の消費電力(20MW)
- 「京」と同程度の設置面積(2000~3000m²)

	総演算性能	総メモリ帯域	総メモリ容量
汎用(従来型)	200~400 PFLOPS	20~40 PB/s	20~40 PB
容量・帯域重視	50~100 PFLOPS	50~100 PB/s	50~100 PB
演算重視	1000~2000 PFLOPS	5~10 PB/s	5~10 PB
メモリ容量削減	500~1000 PFLOPS	250~500 PB/s	0.1~0.2 PB
京(参考)	10 PFLOPS	5 PB/s	1.2 PB



最もエクサフロップスに近いのは演算重視型

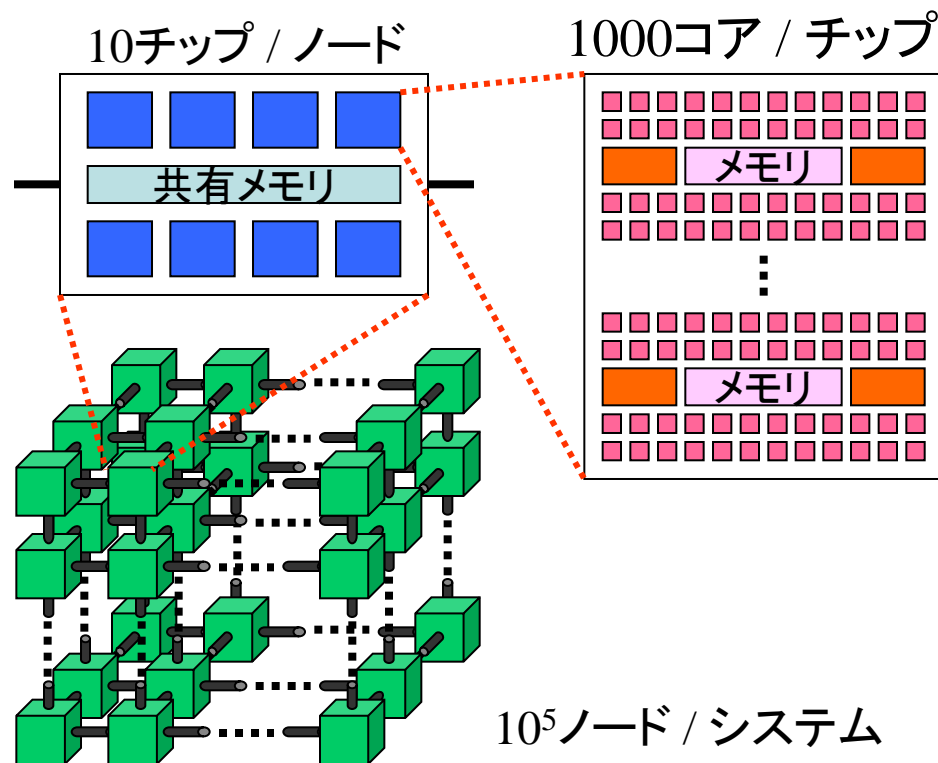
エクサフロップスマシンの特徴(1)

- 多階層の並列性

- 10^9 個程度の演算コア(動作周波数はGHzオーダー)
- コアレベル, チップレベル, ノードレベル, システムレベルの並列性

- 複雑なメモリ階層

- コア毎のレジスタ
- コア毎のキャッシュメモリ
- オンチップメモリ
- ノード内共有メモリ
- 他ノードのメモリ



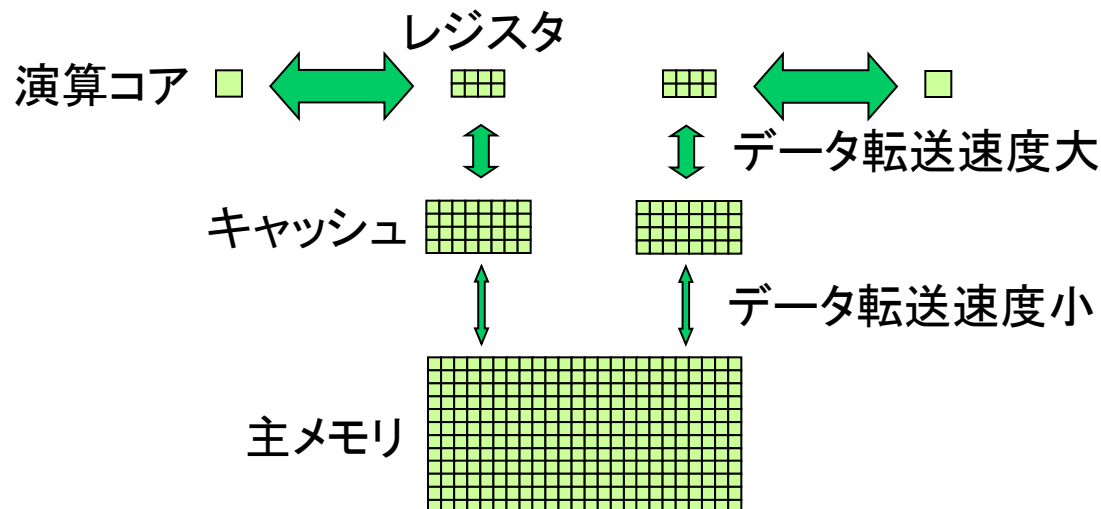
エクサフロップスマシンの特徴(2)

- データ移動コストの増大: スループット

- 主メモリのByte/Flop(データ転送性能と演算性能の比)による比較

	総演算性能	総メモリ帯域	比
演算重視型	1000~2000PFLOPS	5~10PB/s	0.005Byte/Flop
京	10PFLOPS	5PB/s	0.5Byte/Flop

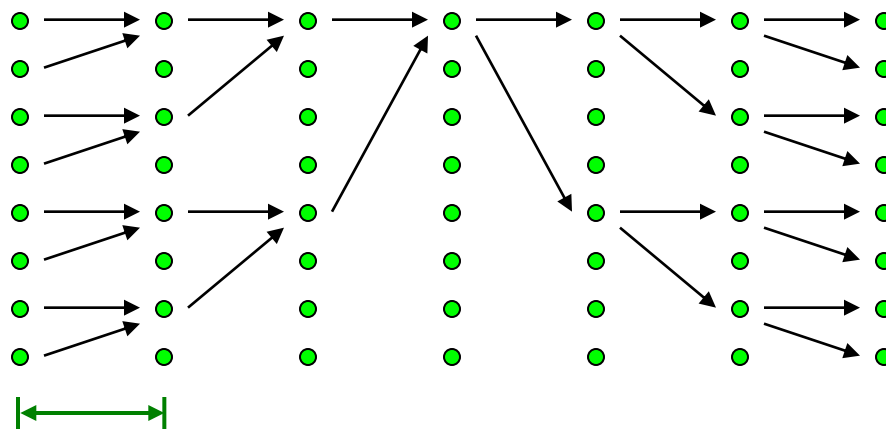
⇒ 主メモリアクセスの相対的成本は、京の場合の100倍



エクサフロップスマシンの特徴(3)

- データ移動コストの増大: レイテンシ(遅延時間)
 - コア間同期・通信レイテンシ: 100サイクル
 - ノード間通信レイテンシ: 80~200サイクル

⇒ AllReduce のような全ノード同期型通信で特に影響大




80~200サイクル

エクサフロップスマシンの特徴(4)

- 電力消費の問題

- 発熱抑制と節電の両面から、電力消費の抑制が重要
- オフチップのデータアクセスが大きな電力を消費
 - 倍精度演算: 1.1pJ/FLOP
 - オンチップデータアクセス: 2.1pJ/Word
 - オフチップデータアクセス: 205pJ/Word

 100倍

- 部品数増加に伴う故障確率上昇

- ハードエラー(熱などによる部品の故障)
- ソフトエラー(α 線などによるビット反転)

ハード/システムソフトレベルでのエラー耐性強化は可能だが、部品数・速度・消費電力の面で大きなコストが必要

アプリケーション並列化における要求の変化

- これまでの並列化研究：弱スケーリングに重点
 - ノード当たりの問題サイズが一定という条件下で並列化
 - ノード数を増やすとともに、問題サイズもどんどん大きくする
 - 比較的、並列性能を出しやすい
- エクサフロップス時代における問題点
 - すべてのアプリで問題サイズ拡大が求められているわけではない
 - 問題サイズは固定し、モデルを精密化したい場合
 - 問題サイズは固定し、時間ステップを増やしたい場合（分子動力学など）
 - 弱スケーリングでは、ノード数増加につれて計算時間も増加
 - 多くの場合、演算量は問題サイズに対して線形より速く増加
 - エクサフロップスマシンの性能を引き出せる問題サイズでは、計算時間が長くなり過ぎて実用的でなくなる場合も

線形計算アルゴリズムの課題(1)

- 10^9 個のコアを活用できる並列性
- データ移動の削減(I): データ移動量の削減
 - データ移動量(階層間/ノード間)に比例してかかるコストの影響を削減

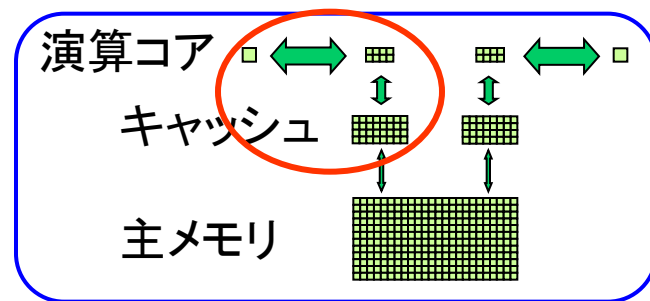


データ再利用性の向上

データが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



消費電力削減の面からも有効



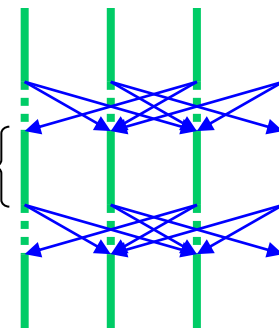
- データ移動の削減(II): データ移動回数の削減
 - データ移動/同期1回ごとにかかるコストの影響を削減



計算粒度の大きいアルゴリズム

同期やデータ移動をできるだけまとめて行う
通信と通信の間で、できるだけ多くの演算を行う

計算粒度



線形計算アルゴリズムの課題(2)

- アルゴリズムレベルでの耐故障性

- 結果不正のノード, 結果を返さないノードがあっても, 計算が破綻せずに進行

- 計算量のオーダーの低減

- ある程度の(確率的)誤差を許容することでオーダーを低減



テンソルの近似に基づくアルゴリズム
確率的アルゴリズム

- 強スケーリングの意味で効率的なアルゴリズム

- 問題サイズを固定したとき, ノード数を増やすほど実行時間が短縮



強スケーリングの条件下では, 通信・同期時間が支配的
大粒度のアルゴリズムはこの場合にも有効

10⁹個のコアを活用できる並列性：密行列の場合

• 正方行列に対するアルゴリズム

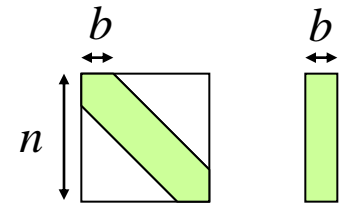
- LU分解, QR分解, 3重対角化, Hessenberg化
- 全体の計算量: $O(n^3)$, 並列性: $O(n^2)$ ⇒ 並列性は十分
- ネックとなりうる部分: ピボット生成, 通信

⇒ これらを隠蔽するための**スケジューリング**が重要

- DAGスケジューリング

• 帯行列・縦長行列に対するアルゴリズム

- 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
- 全体の計算量: $O(n^2b)$ or $O(nb^2)$, 並列性: $O(b^2)$, $O(nb)$ or $O(n)$



⇒ **新たな並列性**を導入できるアルゴリズムが必要

- 帯行列に対する分割統治型のLU分解
- Compact WY 表現に基づくベクトル逐次添加型直交化法

10⁹個のコアを活用できる並列性：疎行列の場合

- 部分空間への射影に基づく反復解法

- 連立1次方程式の解法：Krylov部分空間法全般
- 固有値問題の解法：Lanczos法, Arnoldi法, Jacobi-Davidson法



部分空間の拡大操作は本質的に**逐次的**
並列性：行列ベクトル積の並列度 $O(nz)$

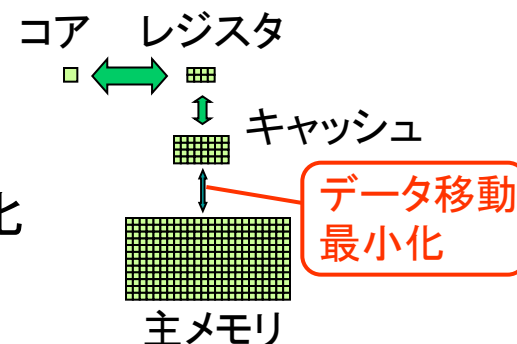
- 並列性を向上させる手法

- ブロックKrylov部分空間法
 - 複数の初期ベクトルから出発し, **複数の部分空間**を同時に生成
 - 部分空間の数だけ並列性が向上
- 数値積分を用いたフィルタで部分空間を生成する手法
 - 櫻井・杉浦法
 - **積分点ごとの並列性**を新たに利用可能

データ移動の削減：密行列の場合

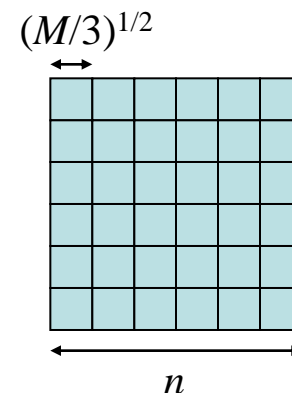
• 想定する状況と目標

- コア: 1個, キャッシュ: M ワード, 行列サイズ: n
- キャッシュと主メモリとの間のデータ移動を最小化
 - ノード間通信最小化でも同様の手法を適用可能



• ブロック化(タイル)アルゴリズム

- 行列を, サイズ $(M/3)^{1/2} \times (M/3)^{1/2}$ のブロックに分割
 - ブロック3個がキャッシュに入る
- 各ブロックを要素と見て行列計算を行う
 - 行列乗算
 - LU分解, コレスキー分解, QR分解
 - ブロック3重対角化, ブロック Hessenberg 化
- 演算の主要部分はブロックどうしの乗算であり, これはキャッシュ上で実行可能



ブロック化アルゴリズムの例

- ブロック化コレスキー分解

- ブロックサイズを $L = (M/3)^{1/2}$, 第 (I, J) ブロックを A_{IJ} とする

```
do K=1, n/L
  AKK := Cholesky(AKK)
  do I=K+1, n/L
    AIK := AIKAKK-T
  end do
  do J=K+1, n/L
    do I=J, n/L
      AIJ := AIK AJKT
    end do
  end do
end do
```

対角ブロックのコレスキー分解

ブロックピボット列の作成

ブロックどうしの行列乗算
(演算の主要部)

- LAPACKで採用されているアルゴリズム

ブロック化アルゴリズムの最適性

- 定理 (Ballard, Demmel, Holtz and Schwartz, 2009)
 - ブロックサイズを $(M/3)^{1/2}$ としたブロック化コレスキー分解は、前記の仮定の下で、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
- 証明
 - 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
 - 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
 - これより、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

\geq

行列乗算における
データ移動量の下界

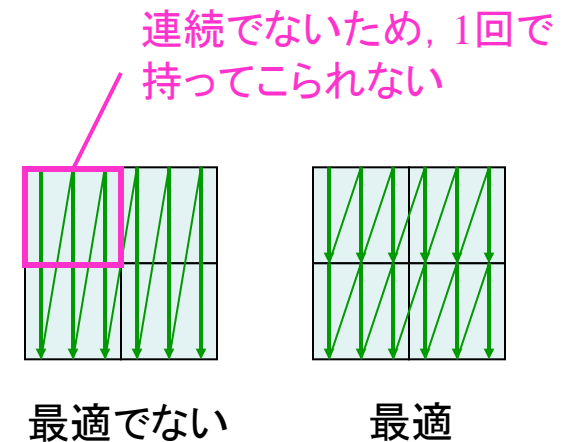
と言える

- ブロック化コレスキー分解が右辺の下界を達成することを示す

Communication optimal なアルゴリズム

- データ移動回数回数の最小化

- 1回のデータ移動では、主メモリの連続した領域しか持ってこれないと仮定
- 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は移動回数最小にならない
- ブロック単位の格納順を使うことで、データ移動回数も最小化



⇒ **Communication optimal** なアルゴリズム

- データ移動量・回数回数の下界が知られているアルゴリズムの例

- 行列乗算 ($O(n^3)$ / Strassen)
- 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

両方について下界を達成する
(Communication optimal な)
アルゴリズムの開発が、活発
な研究テーマ

データ移動の削減：疎行列の場合

- Krylov部分空間法

- $K_k(A; b) = \text{span}\{b, Ab, A^2b, \dots, A^{k-1}b\}$ の中で近似解を求めていく解法
- 演算の主要部分は疎行列ベクトル積 $y = Ax$
- 1ステップ中に複数回の内積(またはノルム計算)が存在

- データ移動の観点からの問題点

- 行列ベクトル積は行列データの再利用性が低い
 - $y = Ax$ において, A の各要素は1回しか計算に利用しない
- 内積は全ノードでの AllReduce を必要とし, レイテンシの影響大

- 課題

- 行列ベクトル積におけるデータ再利用性の向上
- 複数の内積をまとめて同期回数を削減

⇒ 以下ではGMRES法を例として手法を紹介

GMRES法

- 原理

- A をかける操作と直交化を繰り返し、部分空間を広げながら正規直交基底 $\{q_1, q_2, q_3, \dots\}$ を生成
- 各ステップにおいて, $\|r^{(n)}\|_2 = \|Ax^{(n)} - b\|_2$ が最小になるよう解を更新

- アルゴリズム (正規直交基底の生成部分)

```
[Arnoldi process]
```

```
 $q_1 = b / \|b\|_2$ 
```

```
for  $n = 1, 2, 3, \dots$ 
```

```
   $v = Aq_n$ 
```

```
  for  $j = 1$  to  $n$ 
```

```
     $h_{jn} = q_j^T v$ 
```

```
     $v = v - h_{jn}q_j$ 
```

```
  end for
```

```
   $h_{n+1,n} = \|v\|_2$ 
```

```
   $q_{n+1} = v / h_{n+1,n}$ 
```

```
end for
```

} 新たなベクトルの生成 (空間の拡張)

} 直交化

} 正規化

手法(I): ブロック GMRES 法

- アイディア

- 複数 (b 本) の初期ベクトル $R^{(0)} = [r_1^{(0)}, \dots, r_b^{(0)}]$ から出発し, ブロック Krylov 部分空間 $K_b^{(m)}(A; R^{(0)})$ 内で解を求める

- 普通の GMRES 法との比較(1ステップあたり)

	GMRES	ブロックGMRES
演算量(行列ベクトル積)	1 ($y=Ax$)	b ($Y=AX$)
行列データのアクセス回数	1 ($y=Ax$)	1 ($Y=AX$)
同期回数*	1	1

再利用性
 b 倍向上

同期回数
実質 $1/b$

* 直交化に compact WY 表現またはCGS法を用いた場合

- 効果

- 行列アクセス回数・同期回数は同じため, 実行時間増加は b 倍以下
- ブロック Krylov 部分空間を使うことによる収束性向上の効果も存在

⇒ 右辺が複数本の場合は有利. 1本でも高速化できる場合もある

手法 (II) : k -step GMRES 法

• アイディア

- 行列ベクトル積 $A\mathbf{r}^{(m)}$, $A^2\mathbf{r}^{(m)}$..., $A^k\mathbf{r}^{(m)}$ 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

• 普通の GMRES 法との比較 (1ステップあたり)

	GMRES	k -step GMRES
演算量 (行列ベクトル積)	1	$1 + \alpha$
行列データのアクセス回数	1	$1 / k$
同期回数*	1	$1 / k$

再利用性
 k 倍向上

同期回数
 $1 / k$

* 直交化に compact WY 表現または CGS 法を用いた場合

• 効果

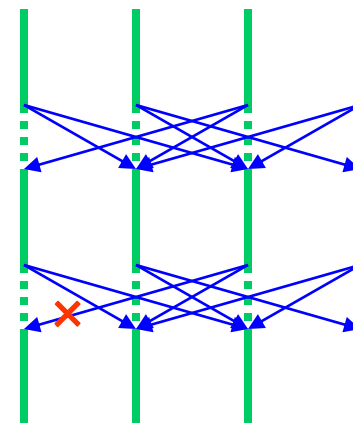
- 再利用性向上と同期の削減により, **大きな性能向上**が期待できる
- ただし, (直交化前の)基底が線形従属に近い場合は, **収束性が悪化**

➡ 線形独立性を高めるため, A の単項式の代わりに直交多項式を使い, 3項間漸化式で計算するなどの工夫が検討されている

アルゴリズムレベルでの耐故障性

• 想定する状況と目標

- 複数のノードが通信を行いつつ協調して計算
- そのうち1個のが不正な結果を返すか、あるいは結果を返さなくても、計算は破綻せずに進行
- その場合、精度劣化、収束性劣化は許容
- 計算の一部を、高信頼モード／高信頼ハードウェア(ただし計算コスト大)で行ってもよい



• 考察

- 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
- 大粒度並列性は、耐故障性にとっても有利
 - 高信頼モードの使用頻度を削減できる

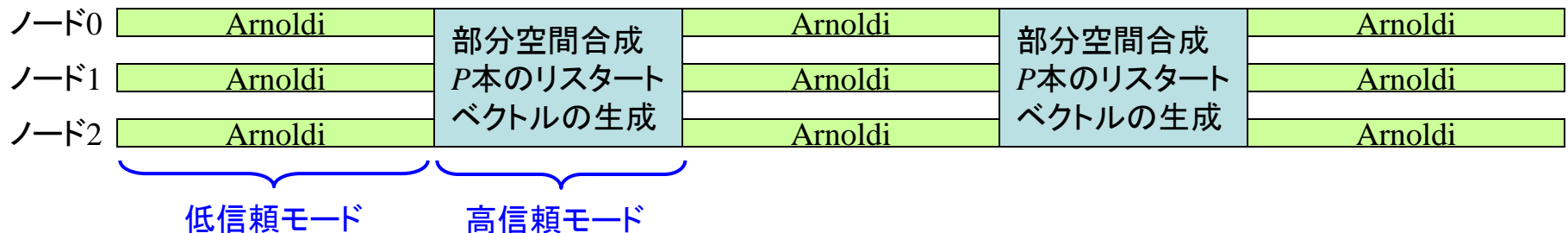
MERAM (Multiple Explicitly Restarted Arnoldi Method)

- アルゴリズム (Emad, Petiton & Edjlali, 2005)

- (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
- (2) 各ノードで作ったKrylov部分空間を合わせ, 大きな部分空間を生成
- (3) その中で P 本の初期ベクトルを新たに生成し, Arnoldi法をリスタート

- 耐故障性

- 上記(1)を低信頼モード, (2), (3)を高信頼モードで実行
 - 計算量の大部分は(1)のステップ
- (1)において, 結果不正のノード / 結果を返さないノードがあっても, 収束性が落ちるだけで, **計算は破綻しない**

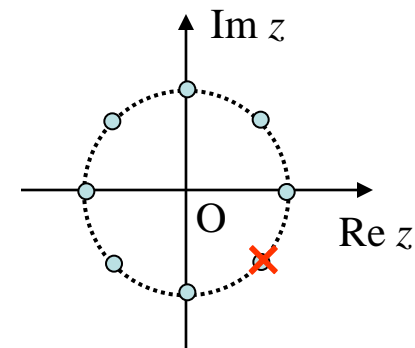


耐故障性を持つその他のアルゴリズム

- Fault-Tolerant GMRES法 (Hoemmen & Heroux, 2011)
 - Flexible GMRES法の枠組みを利用
 - 結果不正を, 不適切な前処理と解釈して計算を続行
 - そのステップでは無駄に部分空間が大きくなるが, 精度には悪影響なし

- 櫻井・杉浦法

- ある積分点を担当するノードが結果を返さない場合, その点を使わない積分公式を新たに構築して使用



- 密行列向けの解法

- 直接解法では, 1箇所の計算間違いで結果に壊滅的な影響
- アルゴリズムレベルでの耐故障性の実現は, 原理的に難しそう

確率的アルゴリズムによる計算量のオーダー低減

- 特異値分解: 行列の最良の低ランク近似を与える分解
 - 画像処理, 信号処理, 情報検索など広い応用
 - $m \times n$ 行列 ($m \geq n$) の場合の演算量は $O(mn^2)$ と大きい
 - CX分解: 特異値分解の代替手法
 - C : 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 : 適当な $k \times n$ 行列
 - このとき, $\|A - CX\|_F$ をできるだけ小さくする C と X を求める
- ⇒ A の特徴を最もよく表す s 本の列ベクトルを選ぶことに相当
- ⇒ Fノルムに対しては, $X = C^+A$ と選ぶのが最適
したがって, 問題は C を選ぶことに帰着

Leverage score に基づく確率的アルゴリズム

- Leverage score (Drineas et al., 2008)
 - A の打ち切り型特異値分解を $A_k = U_k \Sigma_k V_k^T$ とする
 - このとき, 次の量を, V_k^T の第 j 列の Leverage score と呼ぶ

$$p_j = \frac{\| (V_k^T)^{(j)} \|_2^2}{k}$$

- サンプリングと誤差評価
 - 確率 p_j に従い, A の列をサンプリングする
 - このとき, 確率 0.9 以上で次の誤差評価が成り立つ

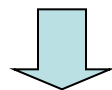
$$\| A - CC^\dagger A \|_F \leq (1 + \epsilon) \| A - A_k \|_F$$

サンプリング 相対評価
回数に依存

- 問題点
 - Leverage score の計算には, V_k^T が必要(このままでは非実用的)

Leverage score の近似

- 近似アルゴリズム (Clarkson et al., 2012)
 - $O(mn \log m)$ の計算量で, Leverage score を近似的に計算するアルゴリズムが存在する
- アイディア
 - A に左から適当な直交行列をかけることで, 一様な Leverage score を持つ行列 A' に変換
 - Johnson & Lindenstrauss の補題と高速 Walsh-Hadamard 変換を使う
 - A' に対して一様なサンプリングを行い, CX 分解を求める
 - A' の CX 分解から, A の CX 分解を求める



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,
 $O(mn \log m)$ で可能に

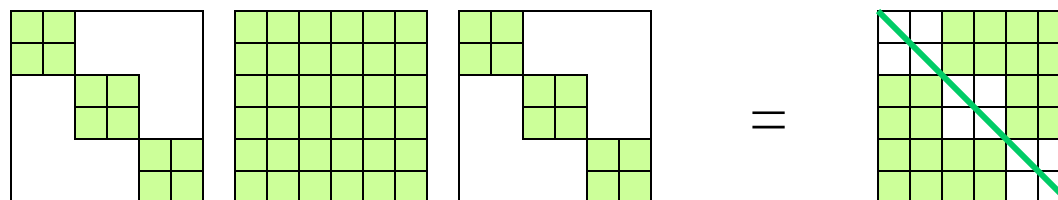
強スケーリングの意味で効率的なアルゴリズム

- 実対称行列の全固有値・固有ベクトル計算
 - 分子軌道法をはじめ, 様々な問題で重要
 - $n=10,000$ 程度の中規模問題に対する需要も多い
- 想定する状況と目標
 - 行列サイズは $n=10,000$ に固定
 - ノードは何個使ってもよいから, できるだけ高速に解きたい
 - 分子軌道法では, 多電子積分の計算量が $O(n^4)$ 以上で並列性も高い
 - この部分を並列化するため, 1万ノード以上を確保している例もある
- 標準的な行列計算ライブラリ(ScaLAPACK)での結果
 - 「京」上では, ノード数を400以上にしても, 計算が加速しない
 - 実行時間の70%以上を占める通信オーバーヘッドのため
 - 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

強スケーリングの意味で効率的なアルゴリズム

- ブロックヤコビ法による全固有値・固有ベクトル計算

- 行列をブロックに分割し, 直交変換により複数の非対角ブロックを並列に消去
- これを繰り返すことで, 行列を対角行列に近づける



- アルゴリズムの特性

- 演算量は3重対角化に基づく解法の10倍以上
- P ノードで2次元分割を行った場合, 通信回数は $O(P^{1/2} \log P * Iter)$

➡ $n=P=10,000$ ならば3重対角化の通信回数 $O(N \log P)$ に比べて小さい通信OHの大きい状況では, 3重対角化法より高速となる可能性

強スケーリングの意味で効率的なアルゴリズム

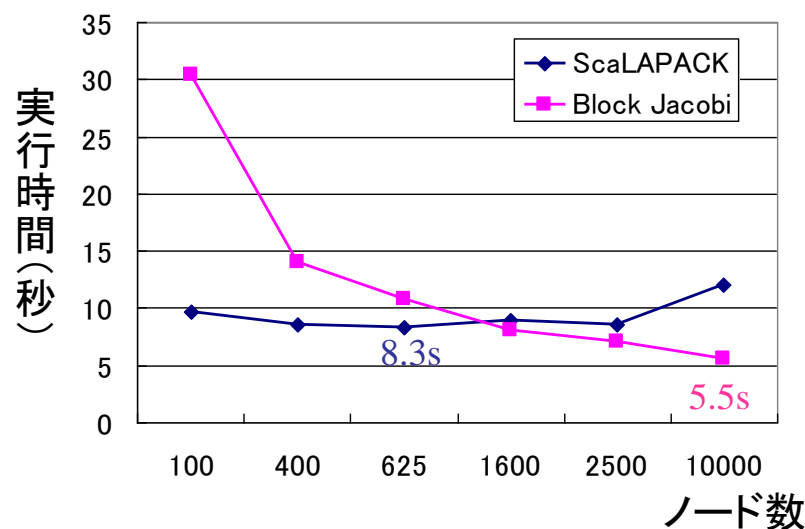
- ScaLAPACK とブロックヤコビ法の性能比較 (Kudo, 2013)

- 「京」上で1万ノードまでを使用して実行時間を測定
- 行列サイズは $n=10,000$ に固定



強スケーリングの条件下では
ブロックヤコビ法が優位

消去順序の最適化, 前処理により,
更に高速化の可能性



- 強スケーリングの条件下でのアルゴリズム設計

- 通信・同期オーバーヘッドの削減が重要
- そのためならば, 演算量のある程度増やしてもよい
 - 中務氏の新しい固有値・特異値計算アルゴリズム

大粒度並列性を持つQR分解手法

- $A \in \mathbf{R}^{m \times n}$ ($m \geq n$) のQR分解

- full QR分解 $A = QR$

$$A = Q R$$

- thin QR分解 $A = \tilde{Q}\tilde{R}$

$$A = \tilde{Q} \tilde{R}$$

- QR分解に対する2つの見方

- \tilde{Q} に着目 \rightarrow A の列ベクトルの正規直交化

- R に着目 \rightarrow 直交行列による A の上三角化

} 様々な応用

- QR分解のアルゴリズム

- ハウスホルダー法

- 古典／修正グラム・シュミット法

} 高性能計算の観点からは課題が多い

CholeskyQR2法による大粒度並列QR分解

- Cholesky QR 法による行列 $X \in \mathbf{R}^{m \times n}$ ($m \geq n$) のQR分解

- アルゴリズム

$$A = X^T X,$$

$$R = \text{Chol}(A). \quad (\text{コレスキー分解: } A = R^T R)$$

$$Y = X R^{-1}.$$

- 長所: **大粒度並列**, すべての計算がブロック化可能
- 短所: X の条件数が大きいとき, 直交行列 Y の直交性が悪化

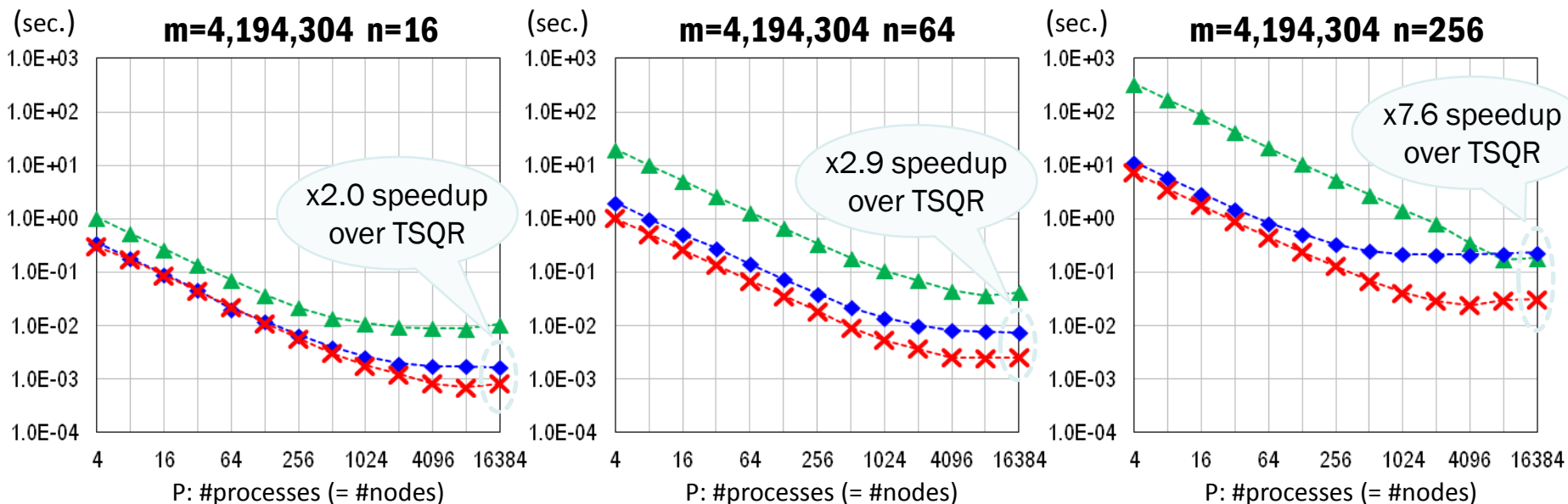
- CholeskyQR2 法^{*1}

- Cholesky QR 法で得られた Y を再直交化
 - Y を X と見て, 上記のアルゴリズムをもう一度繰り返す
- X の条件数が **10^8 以下** ならば, 極めて精度の良い結果が得られる
 - 直交性 $\|Q^T Q - I\|$ も残差 $\|X - YR\|$ も丸め誤差レベル

^{*1} Y. Yamamoto et al.: “Roundoff Error Analysis of the CholeskyQR2 Algorithm”,
Electronic Transactions on Numerical Analysis, Vol. 44, pp. 306-326 (2015).

「京」上でのCholeskyQR2法の性能

---▲--- ScaLAPACK (Householder QR) ---◆--- TSQR ---×--- CholeskyQR2



- TSQRとCholeskyQR2はScaLAPACK(ハウスホルダーQR)より高速
(n=256でPが大きい場合を除く)
- CholeskyQR2はTSQRよりも高速
(Pが大きい場合だけでなく、Pが小さい場合でも)

コレスキーQR分解を2回繰り返してもまだTSQRより高速！

CholeskyQR2法の応用

- 第一原理分子動力学法における波動関数の時間発展

$$m\ddot{\psi}_i(\vec{r}, t) = -\frac{\delta E_{tot}}{\delta \psi_i^*(\vec{r}, t)} + \sum_j \Lambda_{i,j} \psi_j(\vec{r}, t) = -H\psi_i(\vec{r}, t) + \sum_j \Lambda_{i,j} \psi_j(\vec{r}, t)$$

- 時間離散化を行った場合

– $X_t = [\psi_1(\mathbf{r}, t), \dots, \psi_N(\mathbf{r}, t)]$ とおくと,

$$\tilde{X}_{t+\Delta t} = X_t + C\Delta t \quad \text{時間発展}$$

$$X_{t+\Delta t} = \tilde{X}_{t+\Delta t} R \quad \text{直交化(QR分解)}$$

- X_t は直交化されているから, Δt が小さければ, $\tilde{X}_{t+\Delta t}$ も直交に近い
- すなわち, 条件数は小さい(列が直交する場合, 条件数は1)

⇒ CholeskyQR2法が適用可能のはず

おわりに

• 本発表のまとめ

- 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、強スケーリングでの並列化効率がより重視される。
- そのため、線形計算アルゴリズムの研究では次の点が重要になる。
 - 10^9 レベルの並列性
 - 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
 - 強スケーリング性
- これらの方向に沿った最近の研究をいくつか紹介した。

- 会議概要

- 線形代数全般に関する(たぶん)世界最大の国際会議
- Communication-avoiding など, HPCに関連する発表も多い
- 本発表でも, この会議で聞いた様々な講演を参考にした

- 出張報告

- <http://www.na.scitec.kobe-u.ac.jp/~yamamoto/conference.html>