ソフトウェア自動チューニング技術の最新動向
～ マルチコア、ヘテロジニアス、10万並列な環境に向けた新しい最適化技術 ～

東京大学情報基盤センター
片桐孝洋

２００９年６月２６日（金）16時00分～17時00分
第9回ＡＮＳ研究会、
京都大学情報メディアセンター北館3階講習室

本資料の置場

- 以下においてありますので、ご興味があればご利用ください。

http://www.abc-lib.org/MyHTML/paper/ANS9-20090626.pdf

講師紹介

学歴
- 1994年3月国立豊田工業高等専門学校情報工学科卒
  - 研究室配属: 井口健　研究室
  - 準学士論文：QR法における収束の加速とQR分解の高速化
- 1994年4月京都大学工学部情報工学科3年編入
- 1996年3月同卒
  - 研究室配属: 富田 研究室
  - 学士論文 ：分散メモリ型並列計算機によるHouseholder法の性能評価
- 1996年4月東京大学大学院理学系研究科情報科学専攻
- 2001年3月同修了、博士（理学）
  - 研究室配属：金田 研究室
  - 博士論文 ： A Study on Large Scale Eigensolvers for Distributed Memory Parallel Machines

職歴
- 2001年12月～2005年3月科学技術振興機構さきがけプログラム「情報基盤と利用環境」領域 研究者
  - 課題：並列実行環境に依存しない高性能数値計算ライブラリ
- 2002年6月～2007年3月電気通信大学 助手
- 2005年3月～2006年1月米国カリフォルニア大学バークレー校 訪問学者
  - 文部科学省海外先進教育研究実践支援プログラム：高性能コンピューティング研究教育
  - ホスト教授：James Demmel, Katherine Yelick
- 2007年4月～現在 東京大学情報基盤センター特任准教授

講演内容

- 第一部：ソフトウェア自動チューニング概論
  - 数理からみた自動チューニングとは
  - 計算機システムからみた自動チューニングとは
- 第二部：自動チューニング機能付き数値計算ライブラリ
  - 密行列固有値ソルバ関連（Householder三重対角化、QR分解）
  - 疎行列連立一次方程式ソルバ（GMRES法）
- 参考資料（電子配布）
  - 第三部：自動チューニング記述用計算機言語
    - ABCLibScript（日本発／初のＡＴ言語、ＡＴ研究成果物の一つ、ある意味世界の先駆け）
    - 行列積計算に対するABCLibScriptのデモンストレーション
  - 第四部：超並列固有値解析アルゴリズム
    - 並列固有値ソルバ（逆反復法、MRRR法、dqds法）

講演内容

- 第一部：ソフトウェア自動チューニング概論
  - 数理からみた自動チューニングとは
  - 計算機システムからみた自動チューニングとは
- 第二部：自動チューニング機能付き数値計算ライブラリ
  - 密行列固有値ソルバ関連（Householder三重対角化、QR分解）
  - 疎行列連立一次方程式ソルバ（GMRES法）

第一部

ソフトウェア自動チューニング概論

●数理、計算機システム、計算機言語、アプリケーション全般におよぶ先進的最適化技術

自動チューニング研究の普及

- 米国応用数理学（SIAM）の全会員に配送される新聞
  - SIAM NEWS 2008年6月
- 国際会議PP08@アトランタ（2008年3月）のＯＳが特集記事で紹介
- 日本人研究者8名
  - 片桐、黒田、中島（東大基盤センター）、櫻井（日立）、高橋（筑波大）、須田（東大）、今村（電通大）、宮田（名大）
- PP08: Automatic Tuning of High-Performance Numerical Libraries: State of the Art and Open Problems

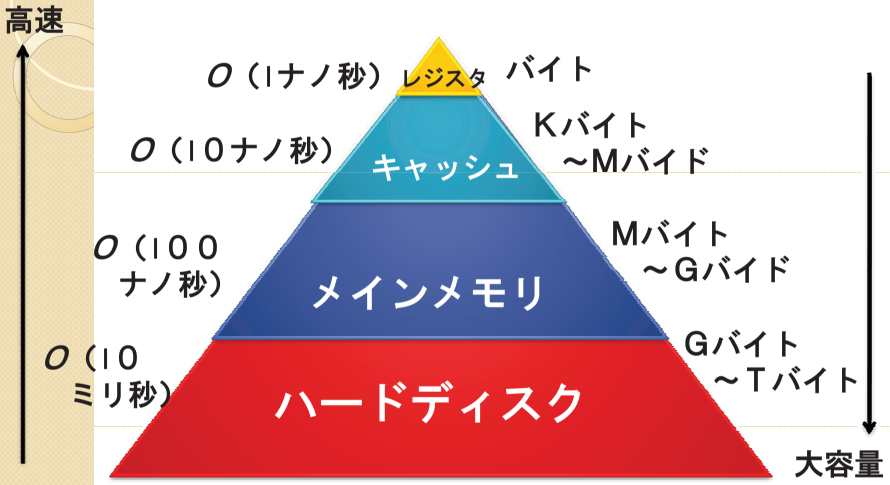
SIAMのWEBページ:
http://sinews.siam.org/old-issues/2008/june-2008/

本題に入る前に

行列計算に必須のコード最適化技法

～　ループアンローリング、ブロック化アルゴリズム、並列化に向くアルゴリズム　～

最近の計算機のメモリ階層構造



•メインメモリ→レジスタへの転送コストは、
レジスタ上のデータアクセスコストの O（100）倍！

ループアンローリングの例（行列-行列積、Fortran言語）

- 普通のコード（IJK-ループ）

```
do i= 1 , n
  do j= 1 , n
    do k= 1 ,  n
      C( i , j ) = C( i ,  j ) + A( i ,  k ) *B( k ,  j )
    enddo
  enddo
enddo
```

- IJKの３重ループは交換できる
  - 6通りの組み合わせ
- ループによりアクセスパターンが違うので性能が異なる
  - どのパターンがよいかは計算機アーキテクチャ依存（実装の違いが＜性能パラメタ＞）

ループアンローリングの例（行列-行列積、Fortran言語）

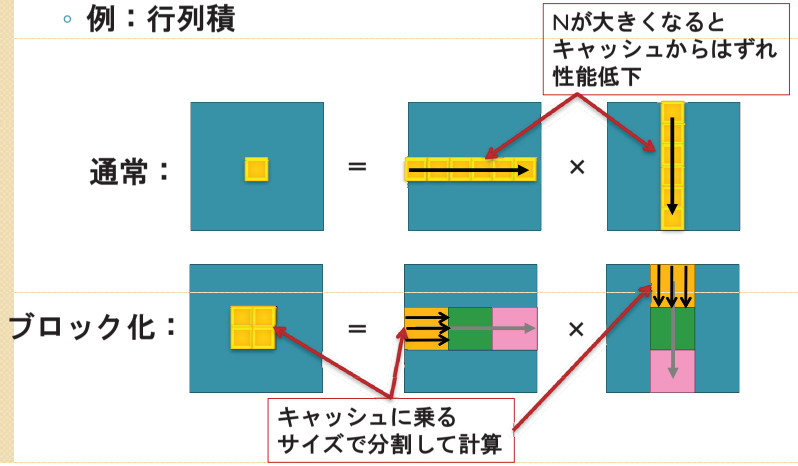
- i-ループ および j-ループ 2段展開
  (nが２で割り切れる場合)

```
do i= 1 , n,  2
  do j= 1 , n,  2
    do k= 1 ,  n
      C(i     , j     ) = C(i     , j     ) +A(i     , k) * B(k, j     )
      C(i     , j+ 1 ) = C(i     , j+ 1 ) +A(i     , k) * B(k, j+ 1 )
      C(i+ 1 , j     ) = C(i+ 1 , j     ) +A(i+ 1 , k) * B(k, j     )
      C(i+ 1 , j+ 1 ) = C(i+ 1 , j+ 1 ) +A(i+ 1 , k) * B(k, j + 1 )
enddo;  enddo;  enddo;
```

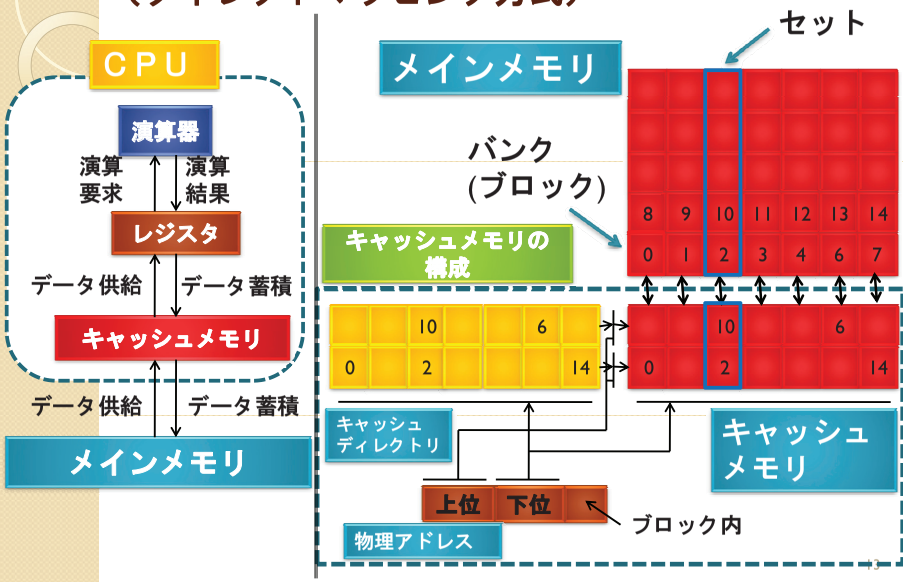
- A(i,j),A(i+１,k),B(k,j),B(k,j+１)をレジスタに置き高速化
- 何段が最適かは計算機依存（段数が＜性能パラメタ＞）

ブロック化とは

- キャッシュを利用するアルゴリズム
- データを局所的に参照するように変更
- データがキャッシュに乗ることで高速化
  - 例：行列積



キャッシュの構成例
（ダイレクトマッピング方式）



行列-行列積のブロック化のコード（Fortran言語）

- nがブロック幅（m=16）で割り切れるとき、以下の6重ループコードになる

```
m = 16          ← ブロック幅（計算機依存＜性能パラメタ＞）
do ib= 1 , n, m
  do jb= 1 , n, m
    do kb= 1 , n, m        ← ブロック幅ごとに進むループ
      do i=ib, ib+m- 1
        do j=jb, jb+m- 1
          do k=kb, kb+m- 1
            C(i, j) = C(i, j) + A(i, k)  *  B(k, j)
enddo; enddo; enddo; enddo; enddo; enddo;
```

ブロック内に局所アクセス化された演算ループ

行列計算との関連

- 先ほどの行列 - 行列積はコード変換レベル
- 行列計算アルゴリズムでは、

変換行列を複数まとめる

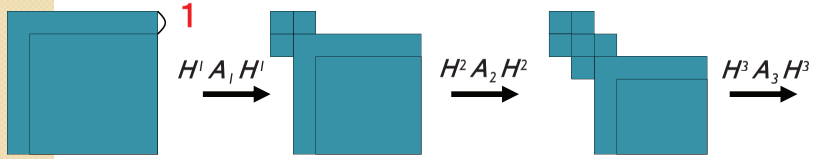
ことで、別アルゴリズムとして実現する

- ＜ブロック化＞アルゴリズム
- LAPACKで採用されている方式

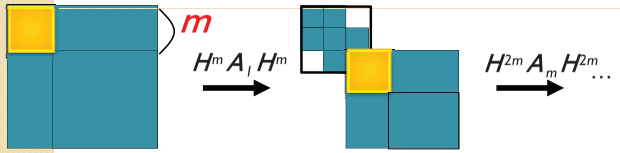
固有値計算におけるブロック化

- Householder変換、QR分解などで利用
- 複数の消去演算をまとめてブロック化
- 主要演算部分：
  BLAS2 (行列 - ベクトル積) → BLAS3 (行列 - 行列積)

通常：



ブロック化： $H^m = H^m H^{m-1} \ldots H^1$



メインメモリアクセス削減手法

- メインメモリ→レジスタへのデータ供給は時間がかかる
- 行列Aのメインメモリからのデータ供給量を減らすため

行列消去演算と、次の消去時の演算、をくっつける（同一ループ中で行う）

方法が古くから知られている。

- 行列消去演算 と 次のステップの行列-ベクトル積 (y=αAu) を 同一ループ中に書く
- 行列Aのメインメモリからのアクセス量が減る
- 先行処理、並列化時にも通信時間が削減
- “Wilkinsonの技巧” [Wilkinson, 65][村上, 2009]

Householder三重対角化のWilkinsonの技巧

- 第 k 反復時の行列 A を $A^{(k)}$と書く

u：枢軸ベクトル

$do \quad k = 1,\ n-2$

$A^{(k)} \rightarrow (u, \alpha)$

$\mu = \alpha u^T x$

$x = \alpha A^{(k)} u$

同一ループ中で行列Aの要素が計算され次第、次のループの行列 - ベクトル積をする

$A^{(k+1)} = A^{(k)} + xu^T + u(\mu u^T - x^T)$

（行列消去演算）

$enddo$

再直交化処理の並列化

- Gram-Schmidtの直交化方式
  - 固有ベクトル ｉ番目を固有ベクトル１～ｉ - １ 番目 $\hat{v}_1, \hat{v}_2, \ldots, \hat{v}_{i-1}$ と＜再＞直交化させるとき

$$v_i = \hat{v}_i - \sum_{k=1}^{i-1} (\hat{v}_k, \hat{v}_i)\hat{v}_k$$

→ 逐次処理では考慮しないが実装方式により並列性が無くなる

再直交化処理の実装（MG-S）

- Modified Gram-Schmidt (修正G-S)法

$t_1 = \hat{v}_i - (\hat{v}_1, \hat{v}_i)\hat{v}_1$

$t_2 = t_1 - (\hat{v}_2, t_1)\hat{v}_2$

…

$v_i = t_{i-1} - (\hat{v}_{i-1}, t_{i-1})\hat{v}_{i-1}$

この式は、明らかに流れ依存がある → 並列化できない

再直交化処理の実装(CG-S)

- Classical Gram-Schmidt (古典G-S) 法

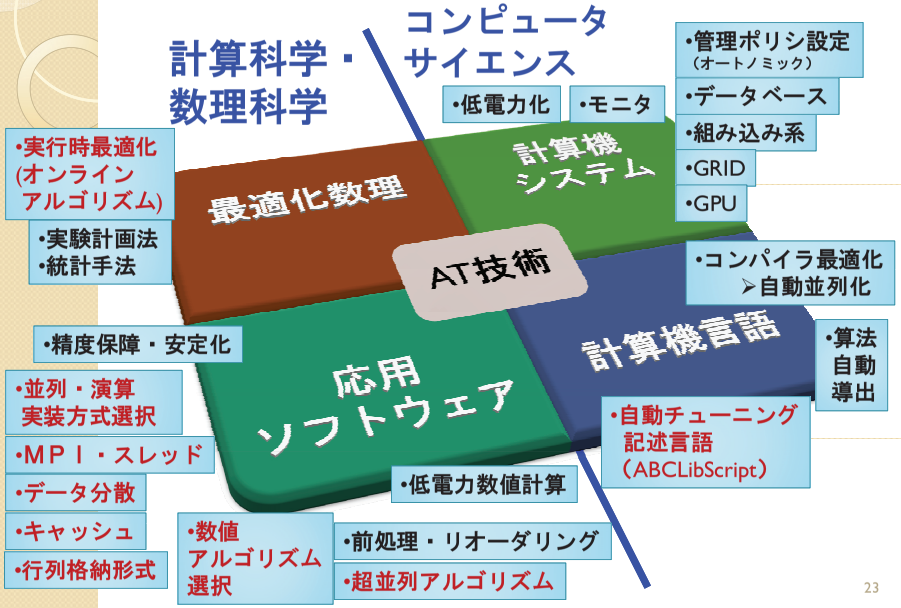
$$t_1 = \hat{v}_i - (\hat{v}_1, \hat{v}_i)\hat{v}_1$$
$$t_2 = -(\hat{v}_2, \hat{v}_i)\hat{v}_2$$
$$\cdots$$
$$t_{i-1} = -(\hat{v}_{i-1}, \hat{v}_i)\hat{v}_{i-1}$$
$$v_i = t_1 + t_2 + \ldots + t_{i-1}$$

$t_1, t_2, \ldots, t_{i-1}$ の計算に関して並列性がある

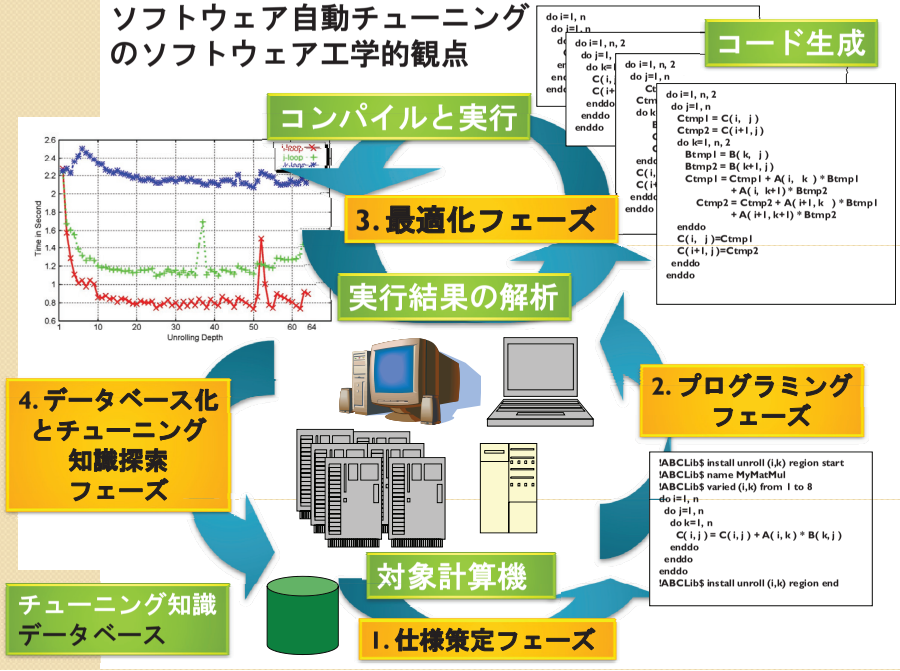
しかし精度の点で修正G-S法に劣る場合がある
→速度と精度のトレードオフ

自動チューニング機構とは

自動チューニング技術の鳥瞰図



ソフトウェア自動チューニングのソフトウェア工学的観点



自動チューニング研究の分類（~2004）

静的チューニング

インストール時

1997 **PHiPAC**（UCB）BLASの自動チューニング

1998 **ATLAS**（U.Tennessee）BLASの自動チューニング

1999 **FFTW**（MIT）FFTの自動チューニング

性能パラメタ定式化

1999 直野ら（日立）による性能パラメタ分類

2001 **SIMPLE**（日立）単一メモリインタフェースによる数値計算ライブラリ

実行起動前時（ユーザ知識によるパラメタの指定後）

動的（実行時）チューニング

数値計算ライブラリ

1998(東大) 疎行列反復解法ライブラリの自動チューニング

計算機システムミドルウェア

1998 **Autopilot**（U Illinois）コンピュータ資源パラメタの動的設定

1999 **Active Harmony**（U. Maryland）コンピュータ資源パラメタの動的設定

1999 **ILIB**（U. Tokyo）自動チューニング機能付き数値計算ライブラリ

融合

2002 FIBER：3つのタイミングによる自動チューニングフレームワーク

2003 **SANS**（U.Tennessee）ライブラリパラメタの自己設定

2004 **ABCLib**（電通大）FIBERに基づく自動チューニング機能付き数値計算ライブラリ

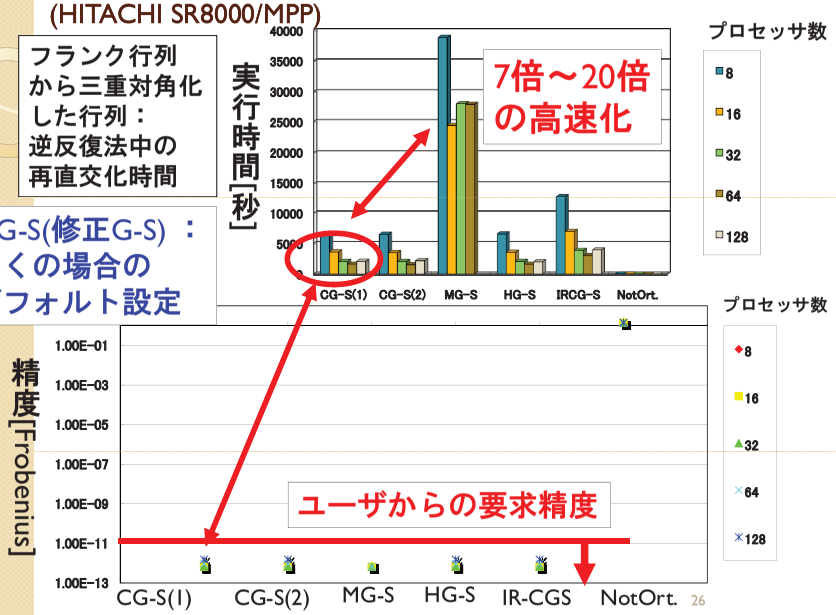
2004 ABCLibScript：自動チューニング記述用言語

実例：実行時アルゴリズム選択

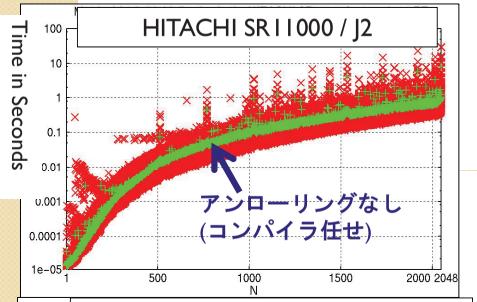
(HITACHI SR8000/MPP)

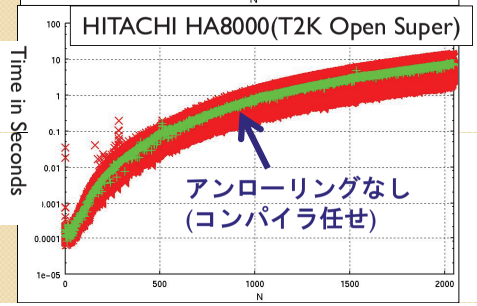
フランク行列から三重対角化した行列：逆反復法中の再直交化時間

MG-S(修正G-S)：多くの場合のデフォルト設定



実例: 先進アーキテクチャによる不安定性





・密行列の行列ー行列積

BLASを用いていない単純コード

・3重ループ (i,j,k)，アンローリング１段～４段

・次元Nに関し　4*4*4=64種類　の実装

・1 から2048次元まで1刻みのデータ.

・コンパイラ HITACHI Optimized　Fortran90. オプション: -Oss

・　自動並列化（ノード内）

・計算機構成:

- HITACHI SR11000/J2
- HA8000 (T2K Open Supercomputer (Todai Combined Cluster))
- Installed in Information Technology Center, The University of Tokyo.
- 16コア／ノード.

●コンパイラ任せは遅い

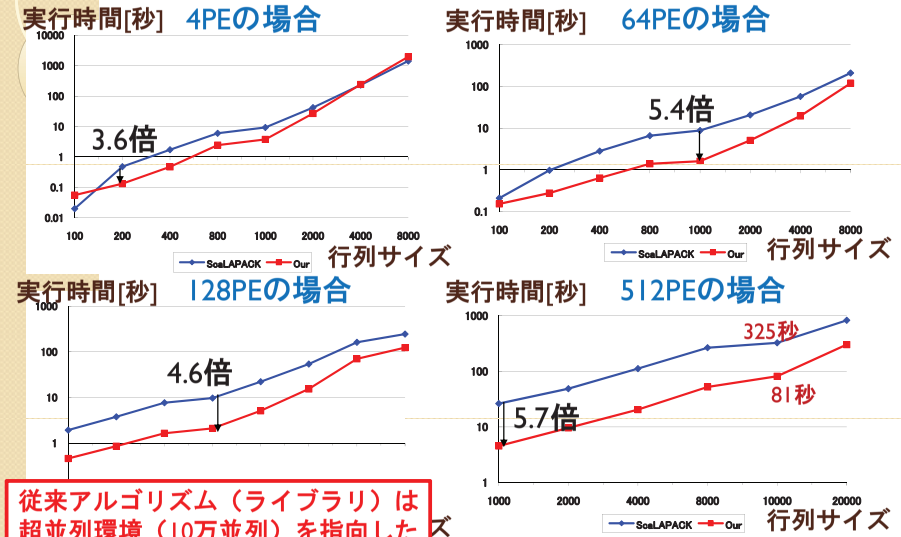
●常時＜特定の実装＞が速くない

●10倍ぐらい実行時間がぶれる（実行時間の＜不安定性＞）

●場合により100倍も遅い！

実例：超並列アルゴリズムの構築と適応的選択

（日立SR2201、Householder三重対角化）



従来アルゴリズム（ライブラリ）は超並列環境（10万並列）を指向したアルゴリズムになっていない

数理からみた自動チューニングとは

- 問題の抽象化
  - ソフトウェアが自分自身を環境に応じて適応させる
  - 環境／ソフトウェア内部に制御変数（チューニングパラメタ）がある
    - パラメタ値は、連続的、離散的どちらもある
  - 試行
    - 実験的な実行
  - 実施
    - 実用的なソフトウェア利用
  - モデル
    - パラメタを変化させると、どのように性能が変化するかの近似関数（目的関数）
  - フィッティング
    - モデルにおける未知変数を試行結果により推定し、最適化すること
- チューニングの数理モデル
  - ソフトウェアの性能を施行もしは実施により測定し、与えられた環境に対し、目的関数を最適にするチューニングパラメタの値を探すこと

須田礼仁：自動チューニングのためのBayes統計に基づく最適化手法、計算工学講演会論文集、Vol.14 、pp.179-182((2009年5月)

数理からみた自動チューニングとは

- Bayes逐次統計に基づくオンライン自動チューニング数理コア
  - オンライン自動チューニング
    - 試行による情報を利用しつつ、実施時に得られる履歴から将来の利用を推定しチューニングを行うこと
  - Bayes統計に基づくモデリング
    - 不確定性の２要因
      1. 撹乱要因による実測のばらつき
      2. 事前情報の不完全性やフィッティングの誤差
    - Bayes統計の「事前分布」でモデルと誤差を表現
    - 実測結果による「事後分布」
    - 以上により、数理的に根拠のある手法の構築
  - 準最適な逐次実験計画法
    - 「bandit problem」に帰着
    - 最適解は計算量が爆発する。少ない計算量で準最適な解を得る方法を提案。
  - 安定したチューニング実現のための条件
    - 「分散成分問題」に帰着し、最適化を達成しない場合がある。
    - 「漸近最適性」
      - （撹乱要因）＞（モデル誤差）の場合モデル誤差が0と誤判定されるが、そうでないならば無限に実施を繰り返す極限で最適になること
    - 「初期実験の有限性」：以下のようにならない性質
      - モデル誤差を過大に大きくすると、最適化そのものができない
      - パラメタの取りうる値が大きすぎると、現実的でない
    - これらの性質を数理レベルで保障する

須田礼仁：自動チューニングのためのBayes統計に基づく最適化手法、計算工学講演会論文集、Vol.14 、pp.179-182(2009年5月)

計算機システムからみた自動チューニングとは

- 利用者（ユーザ）
  - マシン環境のハードウエア性能を、人手を介さず最大限に引き出すことを可能とするソフトウエア技術
- 事業者（ベンダ）
  - 新世代のマシン環境用応用ソフトウエアを、新たに開発することを不要にする方法論
- 技術者
  - プログラムを新しいマシン環境へ移植（写像）するとき、その過程で自動的に最適化を行う技術

弓場敏嗣：数値計算応用を対象とする自動性能チューニング技術、電気通信大学紀要、Vol.20、No.1-2、pp.1-14　(2007)

計算機システムからみた自動チューニングへの技術要求

- コンパイラでできそうなこと
  - アンローリング段数調整
  - タイル（キャッシュ）サイズ調整
- コンパイラでできないこと
  - 数値計算アルゴリズム選択
    - 数値計算精度を保証した上でのアルゴリズム選択
    - 実行時ユーザ知識情報を活用した最適化
- ミドルウエアレベル
  - 通信ライブラリ（MPI）実装方式選択
- 自動チューニングのタイミング
  - インストール時から＜実行時＞へ
    - 以下の方式研究が活発
      - 実行時の問題知識の自動抽出
      - 得られた知識の利用

計算機環境の変化

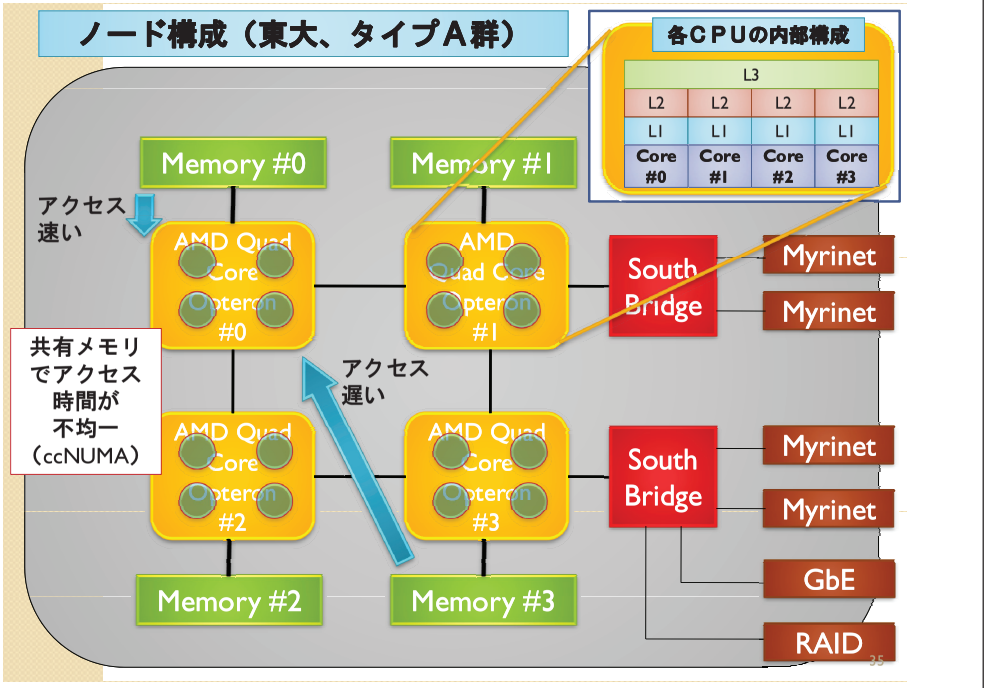
- マルチコアの浸透
  - 非均質メモリアクセス(ccNUMA)
  - 多階層化されたキャッシュ構造
    - L1、L2は局所、L3は共有
  - チップ内のコア数の増大
    - ハイエンド：32コア～、ローエンド：8コア～
- 並列実行モデルの変化
  - ピュアMPI
  - ハイブリッドMPI（OpenMP + MPI）
- コンパイラでは手に負えない
  - 手動チューニングはコスト高
  - コスト削減のため、実用的観点から、性能自動チューニング技術へ期待

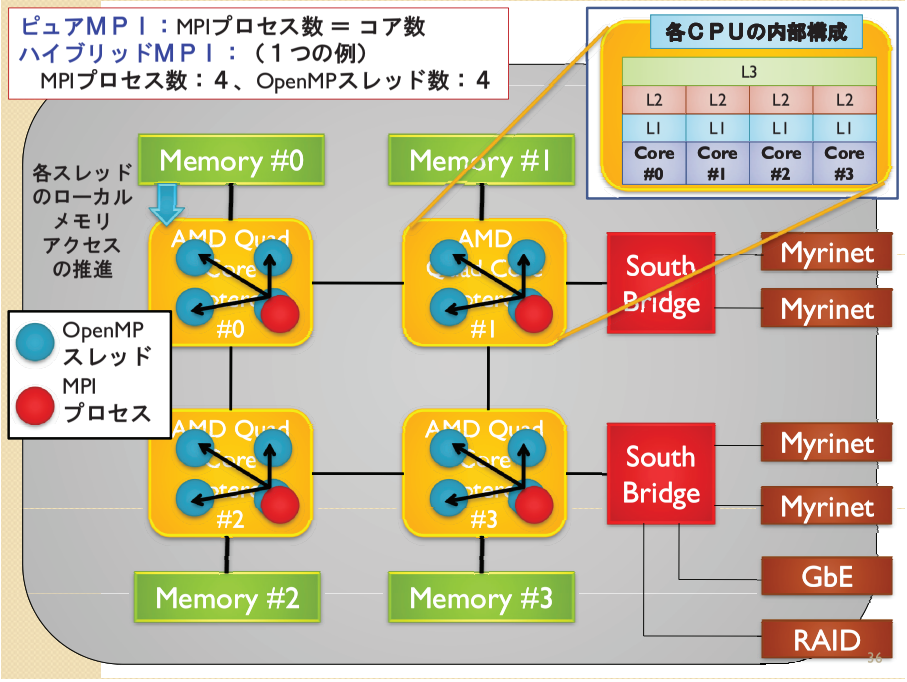
実例：Ｔ２Ｋオープンスパコン ＣＰＵ緒元

（AMD Quad Core Opteron 2.3GHz）

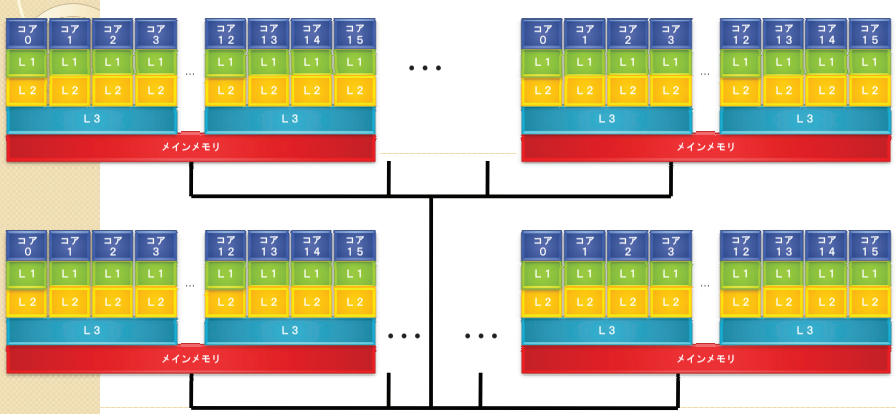
：東大、京大、筑波大に設置のスパコン

項目	値
L1キャッシュ	64 Kbytes（命令、データ、双方は分離） 2 Way Associativity（ライトバック、3サイクル） キャッシュライン：64 bytes、LRU置換
L1データTLB	Full Associativity 2Mbyte Page:8エントリ, 4Kbyte Page:32エントリ
L2キャッシュ	512 Kbytes (2300 MHz)
L3キャッシュ	2048 Kbytes
命令デコード	3 Way
演算実行	3 Way（整数、アドレス生成、浮動小数点 ）
SIMD命令実行	MMX, SSE, SSE2
命令実行	乱発行乱終了(Out-of-order) 整数、浮動小数点
レジスタ	● レガシーモード：汎用32bit: 8個、128bit-XMM:8個、64bit MMX:8個（x87互換用:8個と同一） ● 64bitモード：汎用64bit:16個、128bit-XMM:16個、64bit MMX:8個（x87互換用:8個と同一）
システムバス	1000 MHz





スパコンでの全体メモリ構成図



メモリが多段に階層化
（L1、L2、L3、分散メモリ）

高速ネットワーク
（５Ｇバイト／秒
×双方向）
（タイプＡ群）

講演内容

- 第一部：ソフトウェア自動チューニング概論
  - 数理からみた自動チューニングとは
  - 計算機システムからみた自動チューニングとは
- 第二部：自動チューニング機能付き数値計算ライブラリ
  - 密行列固有値ソルバ関連（Householder三重対角化、QR分解）
  - 疎行列連立一次方程式ソルバ（GMRES法）

第二部

自動チューニング機能付き数値計算ライブラリ

●チューニング作業不要な
先進的数値計算ライブラリ

密行列固有値ソルバ ABCLIB_DRSSED

固有値問題とは

- 標準固有値問題

$$Ax = \lambda x$$

x：固有ベクトル　λ：固有値

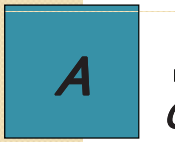
- 応用分野
  - 科学技術計算分野全般
  - 量子化学分野（密行列、大部分の固有値・固有ベクトル）
  - インターネット検索（知識発見）[M.Berryら, 1995]
- 密行列：$O(n^3)$ の計算量
  - 高速化（効率的な逐次実装、並列化）が必須

固有値計算の古典的逐次アルゴリズム

（標準固有値問題 $Ax = \lambda x$ ）

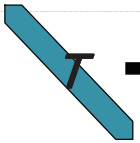
対称実数密行列

1. Householder 変換



$QAQ=T$

三重対角化



三重対角行列

2. ２分法

三重対角行列Tの **全固有値：Λ**

3. 逆反復法

三重対角行列Tの **全固有ベクトル：Y**

4. Householder逆変換

元の対称行列Aの **全固有ベクトル：X = QY**

ベンチマークとしての特徴

- 対称実数固有値ソルバ（ブロック化なし）
  - Householder三重対角化部（以降，TRD）
    - BLAS 2演算性能評価
      - 行列-ベクトル積演算性能
      - 行列更新演算性能
    - マルチキャスト通信性能評価
      - プロセッサのグリッド（p×q）における，同時放送処理 p 個（従事プロセッサ数 q 個）の性能
  - Householder逆変換部（以降，HIT）
    - BLAS 1演算性能評価
      - 必要な固有ベクトル数（k）に応じて BLAS 1-k更新となる行列更新演算性能．全固有ベクトルが必要なときは、BLAS 2演算となる．
    - 集団通信性能評価
      - Gather演算性能
- QR分解（ブロック化あり）注：再直交化ではない
  - 修正Gram-Schmidt演算部（以降，MGSAO）
    - BLAS3演算性能評価
      - 行列更新演算性能．ただし，最外ループの刻み幅はブロック幅となる．
    - 1 対 1 通信性能評価
      - 通信粒度がブロック幅の逆数で変化する．
      - 通信回数が，上記BLAS3のブロック幅の逆数となる実装における評価．
- 両方とも、ベクトル（疑似ベクトル）計算機向き
  - 最内側ループ長が長い

並列Householder三重対角化アルゴリズム

３つの基本演算部分

```
<1>  do k=1, n-2
<2>    if (k ∈ Γ ) then
<3>      broadcast( A(k)_{Π,k} ) to PE sharing
           rows Π
<4>    else
<5>      receive( A(k)_{Π,k} )
<6>    endif
<7>    computation of (u_Π, α )
<8>    if (I have diagonal elements of A)
         then
<9>      broadcast(u_Π) to PEs sharing
           columns Γ
<10>   else
<11>     receive( u_Γ )
<12>   endif
<13>   do j=k, n
<14>     if (j ∈ Γ) u_Π = u_Π + α A(k)_{Π,j} v_j
<15>   enddo
<16>   global summation of u_Π to PEs
         sharing rows Π
```

(1) CPU間の加算

(3) 行列更新

(2) 行列-ベクトル積

```
<17>   if (I have diagonal elements of A)
         then
<18>     broadcast( u_Π ) to PEs sharing
           columns Γ
<19>   else
<20>     receive( u_Γ )
<21>   endif
<22>   do j=k, n
<23>     μ = α u_Π^T x_Π enddo
<24>   global summation of μ to
         PEs sharing rows Π
<25>   do j=k, n
<26>     do i=k, n
<27>       if (i ∈ Π .and. j ∈ Γ ) then
<28>   A(k+1)_{i,j} = A(k+1)_{i,j} − ν_i(χ_j^T − μ v_j^T) − χ_i v_j^T
<29>       endif enddo enddo
<30>   if (k ∈ Γ) Γ = Γ − {k} endif
<31>   if (k ∈ Π) Π = Π − {k} endif
<32> enddo
```

並列Householder逆変換アルゴリズム

<1> do k=1, n
<2> Gather the vector u_i and scalar α_i by using row-wise multi-casting.
<3> Computation of σ_i
<4> do k=kstart, kend
<5> $w_k = w_k - \sigma_i u_i$
<6> enddo
<7> enddo

(1) メインカーネル

並列MGS直交化アルゴリズム

<1> do k=1, n, im
<2> if ($k \in \Gamma$) then
<3> do j=k, k+im-1
<4> Normalization of w_j
<5> $w_j^{(k-1)} = w_j$
<6> do i=k, j+1
<7> $w_j^{(i)} = w_j^{(i-1)} - (w_i^T, w_j^{(i-1)}) w_i$
<8> enddo
<9> $w_{j+1} = w_j^{(j+1)}$
<10> enddo

(1) 枢軸PEカーネル

<11> broadcast($w_k, \ldots, w_{k+im-1}$)
<12> jstart=jstart+1
<13> else
<14> receive($w_k, \ldots, w_{k+im-1}$)
<15> do j=jstart, n
<16> $w_j^{(k-1)} = w_j$
<17> do i=k, j+j-jstart+1
<18> $w_j^{(i)} = w_j^{(i-1)} - (w_i^T, w_j^{(i-1)}) w_i$
<19> enddo
<20> enddo
<21> endif
<22> if ($k \in \Gamma$) $\Gamma = \Gamma - \{k\}$
endif
<23> enddo

(2) 内部PEカーネル

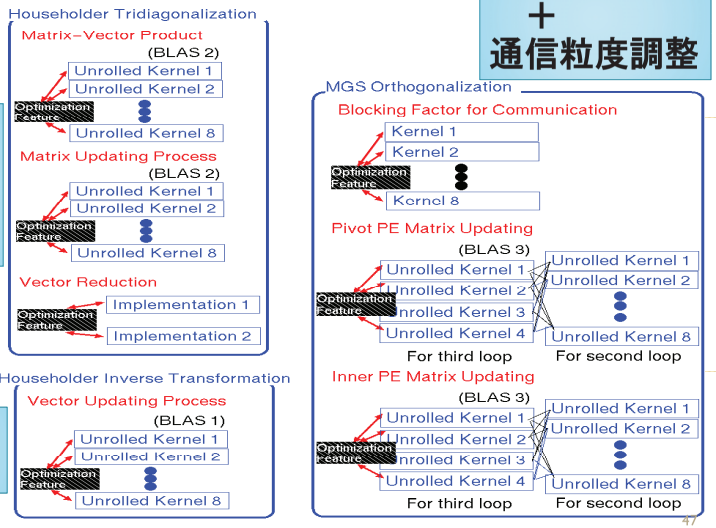
自動チューニング機構
(ABCLib_DRSSED ver.1.04)

全コード量：約20,000行
（Fortran90、MPI-1）

BLAS3演算
＋
通信粒度調整

BLAS2演算
＋
通信実装
方式選択

BLAS 1-k
演算



自動チューニング空間

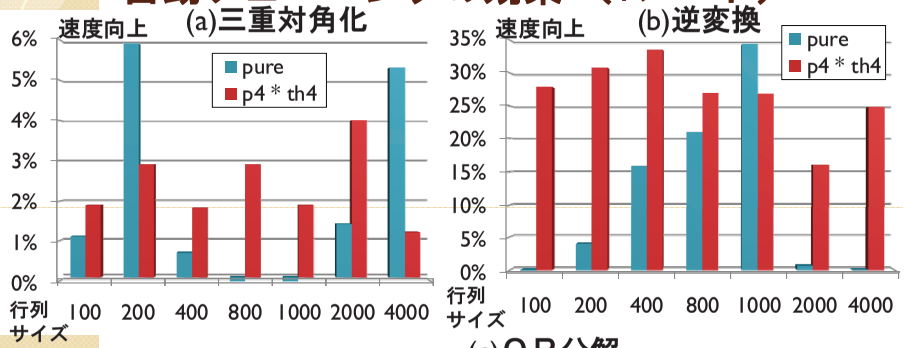
- 対称実数固有値ソルバ　性能パラメタ定義域
  - Householder三重対角化用（16×2+2=34実装／サンプリング点）
    - 通信実装方式切り替え
      ictr = {1対1通信関数を用いた実装，集団通信関数を用いた実装}
    - 行列-ベクトル積アンローリング段数制御
      imv = {1段,2段,…,16段}：最外ループ
    - 行列更新演算アンローリング段数制御
      iud = {1段,2段,…,16段}：最外ループ
  - Householder逆変換用（16+3=19実装／サンプリング点）
    - 通信実装方式切り替え
      ichit = {放送関数を用いた実装，1対1ブロッキング通信関数を用いた実装，1対1ノンブロッキング関数を用いた実装}
    - 行列更新演算アンローリング段数制御
      ihit = {1段,2段,…,16段}：最外ループ
- QR分解　性能パラメタ定義域（6+4*6*2=54実装／サンプリング点）
  - ブロック幅調整
    ibl = {1,2,3,4,8,16}
  - 枢軸ブロック演算用
    - 最外ループアンローリング段数制御
      iop = {1段,2段,3段,4段}
    - 第２ループアンローリング段数制御
      isp = {1段,2段,3段,4段,8段,16段}
  - 主演算用
    - 最外ループアンローリング段数制御
      ioo = {1段,2段,3段,4段}
    - 第２ループアンローリング段数制御
      iso = {1段,2段,3段,4段,8段,16段}
- 以上のパラメタ空間を全自動で全探索し、**最適点を設定**

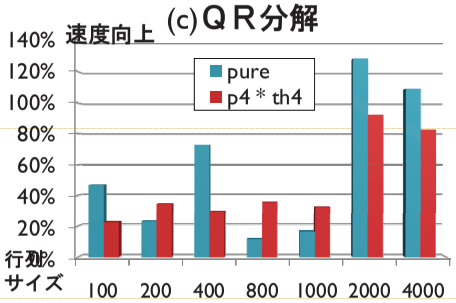
自動チューニングのインパクト

1. 今回の全探索空間の概略
   - サンプリング点（行列サイズ）＝8点
   - 固有値ソルバ＝53×8＝424点
   - QR分解＝54×8＝432点
   - 以上合計：856点／実行ノード
   - さらに、ピュアＭＰＩとハイブリッドＭＰＩで２実装なので、
     総合は1,712点／実行ノード
2. 各点における実行時間は非均一（問題サイズ依存）
   - バッチジョブ制約（東大）：実行ノードを固定として、実行時間は１回あたり２４時間が上限
   - 性能評価のためのデータ採取期間：～１週間

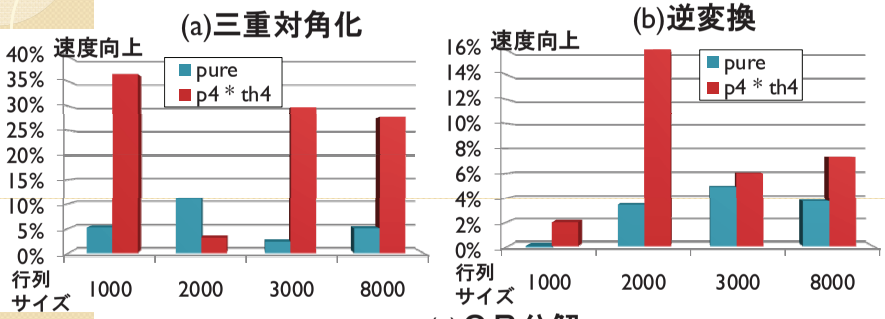
- 以上の１、２を考慮すると、人手では性能評価すら困難（手間／時間的に）

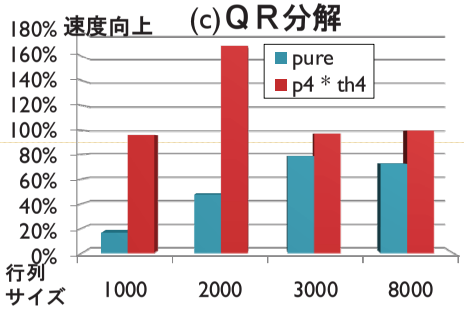
自動チューニングの効果 （1ノード）



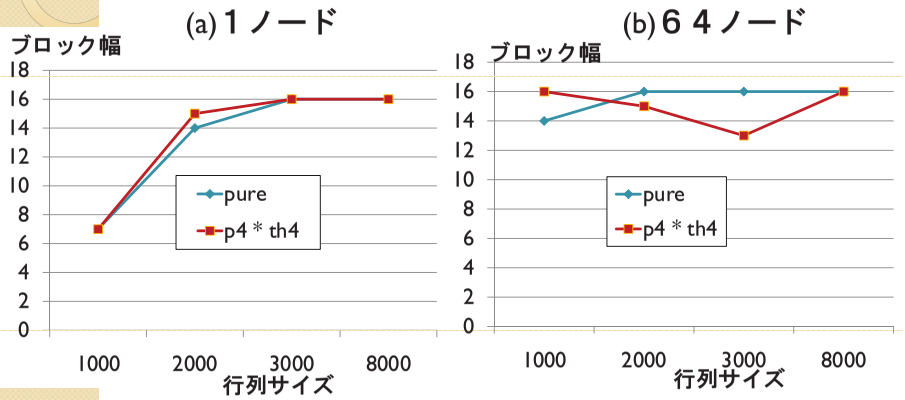


自動チューニングの効果 （64ノード）

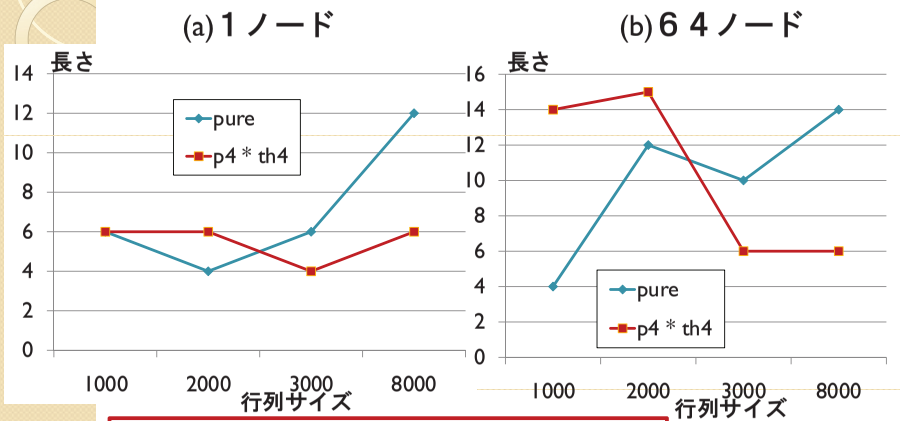




選択実装方式の変化：最適ブロック幅 （ＱＲ分解）



選択実装方式の変化：アンローリング段数
（ＱＲ分解、指標：＜最外ループ段数×第2ループ段数＞）

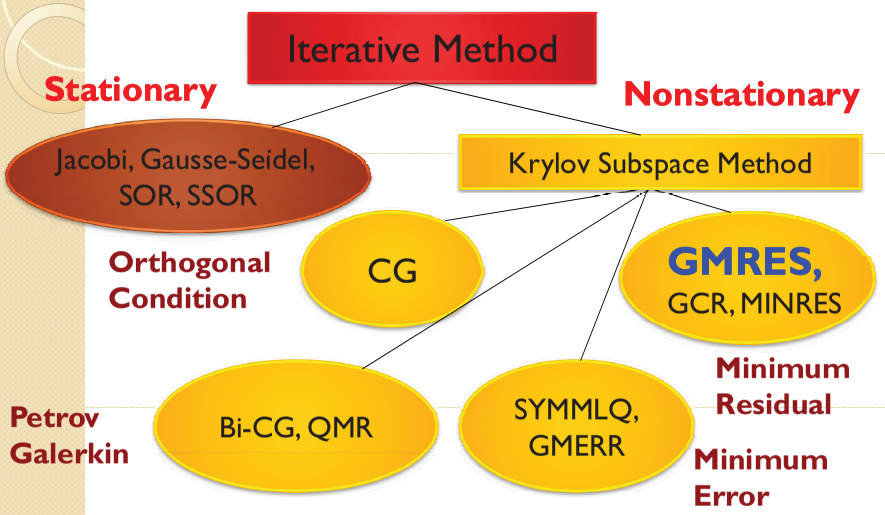


ピュアMPIとハイブリッドMPIで
全く異なる実装が最適。
この選択も、自動的に可能！

疎行列連立一次方程式 ソルバ ILIB_GMRES (MPI並列化版)

※愛媛大学　黒田久泰先生
との共同研究

Iterative Algorithms for Linear Equations



Why Sparse Iterative Methods?

$$Ax = b$$

A is a sparse matrix. x, b are dense vectors.

- Many problems on scientific & engineering field are formulized.
- Direct solving (LU) is not effective.
  - Viewpoint of memory and computation complexity.
    - Increase fill-in elements.
  - Use iterative algorithm to save memory space.
- **Problems:**
  1. Convergence (Preconditioner, Restart Frequency)
     - Setting algorithm parameters needs numerical background.
  2. Computational Efficiency
     - Sparse Matrix-Vector Products (SpMxV)
  3. Efficient Parallel Implementation

GMRES(*m*) Algorithm Features

- Robust Algorithm
  - Reduce the norm of residual vector according to iteration count, theoretically.
  - Applied to many applications.
- Setting Restart Frequency "*m*" Is Difficult
  - Large *m* makes good convergence in general.
  - **Large *m* damages memory and computation order: Memory : O (*mn*) , Computational : O (*nm²*)**
  - Frequently Used Manner for the *m* :
    1. Use Library Default Value (Like *m*=30)
       - It does not converge in some cases.
    2. Maximize *m* to memory restriction
       - It can converge, however it takes much time.
       - The time of orthogonalization is increased, since it is not parallelized (depends on algorithm).

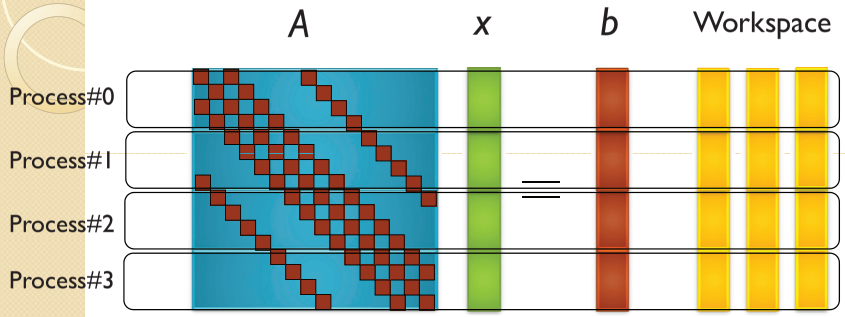
The GMRES(*m*) Details

- Linear systems for non-symmetric matrix
- Krylov subspace method with uses the minimum residual approach at every iteration step
- Restarts the iteration each m steps

x_{start} = 0
loop:
for i=1, 2, ···, m (m: restart frequency)
  w = $Av^{(i)}$
  for j=1, 2, ···, i
    $h_{j,i}=(w, v^{(j)})$
    $w = w - h_{j,i}v^{(i)}$
  enddo
  $h_{i+1,i}$ = ‖w‖ ; $v^{(i+1)}=w / \|w\|$
enddo
solve Hy = $\|r\| e_1$
$x = x_{start}+y_1v^{(1)}+y_2v^{(2)}+y_3v^{(3)}+\cdots+y_iv^{(i)}$
tolerance check
$x_{start}=x$
goto loop:

Communication required
Sparse matrix by vector product
Orthogonalization of i vectors
Every processor performs the same computation
QR decomposition
Decide y to minimize ‖ b − $Ax^{(i)}$ ‖

Inner Data Formant for SpMxV

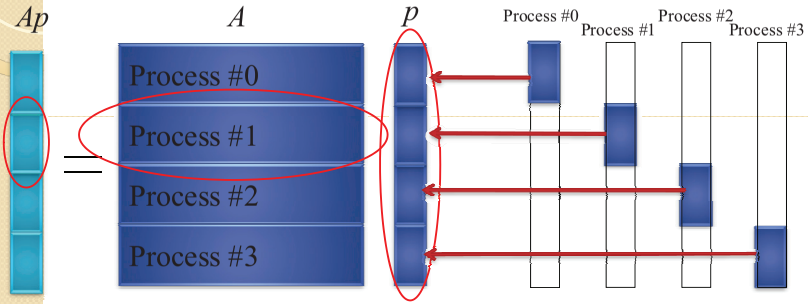


Matrix *A* is distributed as row manner.

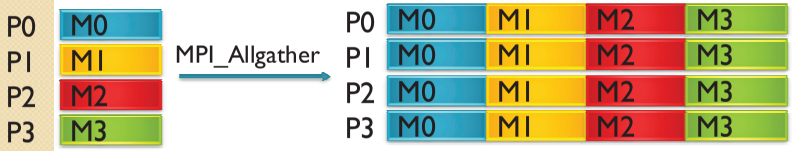
- Matrix *A* is only saved the elements with their indexes.
  - c.f. CRS (Compressed Row Storage)

Vectors *x* and *b* and workspaces are distributed as block manner.

Simple Implementation of Parallel SpMxV : Using MPI_Allgather Function



- MPI_Allgather works:



An Example of Parallel SpMxV Code

- SpMxV ($q=Ap$)

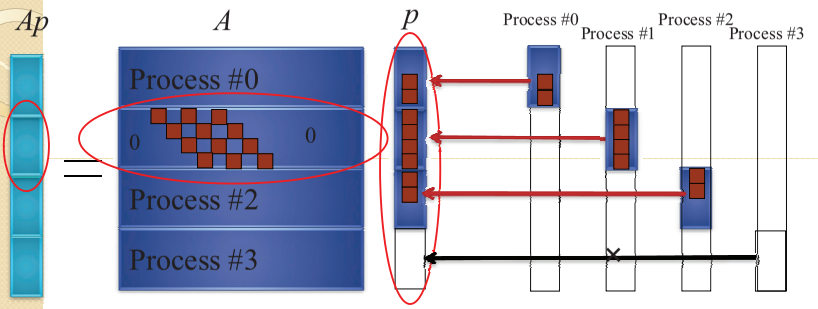
```
void spmat_vec(double q[], double a[], int ir[], int ic[],
               double p[], int localsize)
{
  MPI_Allgather(&p[rank*localsize], localsize, MPI_DOUBLE,
               p, localsize, MPI_DOUBLE, MPI_COMM_WORLD);

  for(i=0;i<localsize;i++){
    q[i]=0
    for(j=ir[i];j<ir[i+1];j++){
        q[i] += a[j]*p[ic[j]];
    }
  }
}
```

To have whole elements of p for all processes

"Serial" SpMxV do as parallel

Reduced Communication Implementation of SpMxV



Whole Elements are not needed.

$$0\cdot p_1+0\cdot p_2+0\cdot p_3+a_4\cdot p_4+a_5\cdot p_5+\cdots$$

↑In this case, we only know the values of p_4 and p_5.

- 1-to-1 communication, like MPI_Send and MPI_Recv, can use to speedup the SpMxV.

Overview of ILIB_GMRES

- **Target Application**
  - **ILIB_GMRES:** A Parallel Sparse Iterative Solver with the GMRES(m) Method
  - Developed by Prof. H.Kuroda
  - Followings are auto-tuned **at run-time** according to input matrix:
    1. **Selection of Preconditioners (Scaling, Block ILU, Polynomial)**
    2. **Adjustment of loop unrolling depth of SpMxV**
    3. **Selection of MPI implementations (Gather, Collectives, 1-to-1 (Blocking or Non-blocking))**

ILIB_GMRES Interface

User source code

Users need not write parallel codes neither use MPI routines

Same as PETSc

ILIB_GMRES routine

int ILIB_GMRES (Mat *A,Vec *x, Vec *b, int max_iter, double tol);

A (input)	:	Coefficient matrix
x (output)	:	Answer vector
b (input)	:	Right-hand side vector
max_time (input)	:	Maximum allowed time
tol (input)	:	Tolerance

Read File operation

MatrixMarket File Format
Harwell/Boeing File Format

IlibInitialize()
IlibFinalize()
IlibGetSize()
IlibGetRank()

Vector operation

VecCreate()
VecDupulicate()
VecGetOwnershipRange()
VecSetValues()
VecSetValue()
VecAssemblyBegin()
VecAssemblyEnd()
VecZeroEntries()
VecDestroy()
VecView()

Matrix operation

MatCreate()
MatGetOwnershipRange()
MatSetValues()
MatSetValue()
MatAssemblyBegin()
MatAssemblyEnd()
MatZeroEntries()
MatDestroy()

Run-time Auto-tuning Facility of ILIB_GMRES

Run-time Auto-tuning Contents:

Implementation

1. Sparse Data Format (corresponding to register blocking)
2. Loop Unrolled Codes for SpMxV
3. Communication Implementations for SpMxV
   (1) MPI_Allgather
   (2) MPI_Bcast
   (3) MPI_Send / Recv (1-to-1 blocking)
   (4) MPI_Isend / Irecv (1-to-1 non-blocking)
   (5) MPI_Irecv / Isend (1-to-1 non-blocking)

Using Communication Tables To Reduce Communication

Algorithmic

4. Re-start Frequency

5. Re-Orthogonalization Algorithm (MGS or CGS)
6. Preconditioners (Scaling, Block ILU, Matrix Polynomial)

Default : No Loop Unrolling, Communication: 1-to-1 Blocking, Scaling, CGS Ort., Restart Frequency: 30 fixed.

The Test Environment

Target

- The effect of AT for restart frequency on ILIB_GMRES
- Default Restart Frequency : 30(same as PETSc)
  ILIB_GMRES:Auto

Computers

T2K Open Supercomputer (Todai) : 4 Nodes

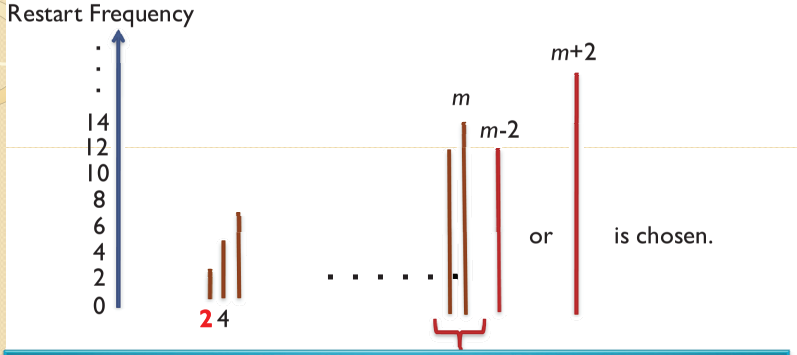
- Each node has 4 Opteron Quad-core Processor 2.3GHz
- Myrinet(Mirycom) full-bisection 2.5GB/s(one direction) interconnection
- Total peak performance of 4 nodes is 588.8GFlops

Problems

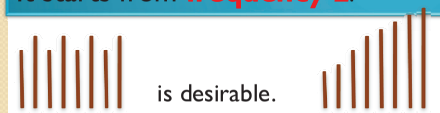
- 3 Matrices from UF Sparse Matrix Collection
  - Difficult cases to solve using GMRES.

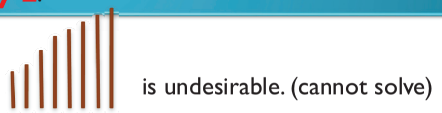
Matrix Name	Size	Nonzeros	minimum	maximum	RHS
chem_master1	40,401	201,201	3	5	$A^*(1,\cdots,1)^T$
epb3	84,617	463,625	3	7	$A^*(1,\cdots,1)^T$
matrix_9	103,430	2,121,550	13	677	$A^*(1,\cdots,1)^T$

A Heuristics Approach: Auto-tuning of Restart Frequency [Kuroda et. al., 2000]



Refers the last two restart frequencies with elapsed time and rate of diminution in residual error.
It starts from **frequency 2**.





Chem_master1 on The T2K

4 cores

Restart	Iter.	Orth.	Time
30	6752	7.74	10.59
40	-	-	-
50	4265	8.19	10.07
60	3816	8.90	10.83
70	3450	9.05	10.39
Auto	5082	5.92	8.19

8 cores

Iter.	Orth.	Time
5914	3.26	4.59
-	-	-
4105	4.28	5.08
3039	3.69	4.35
3337	4.96	5.60
3453	2.86	3.73

16 cores

Iter.	Orth.	Time
4607	0.77	1.79
-	-	-
3579	1.22	1.87
3997	2.20	2.95
3755	3.21	3.98
3672	1.17	1.84

32 cores (2 nodes)

Restart	Iter.	Orth.	Time
30	5306	0.71	1.53
40	-	-	-
50	3974	0.71	1.30
60	3632	0.72	1.36
70	3566	0.83	1.42
Auto	3818	0.60	1.61

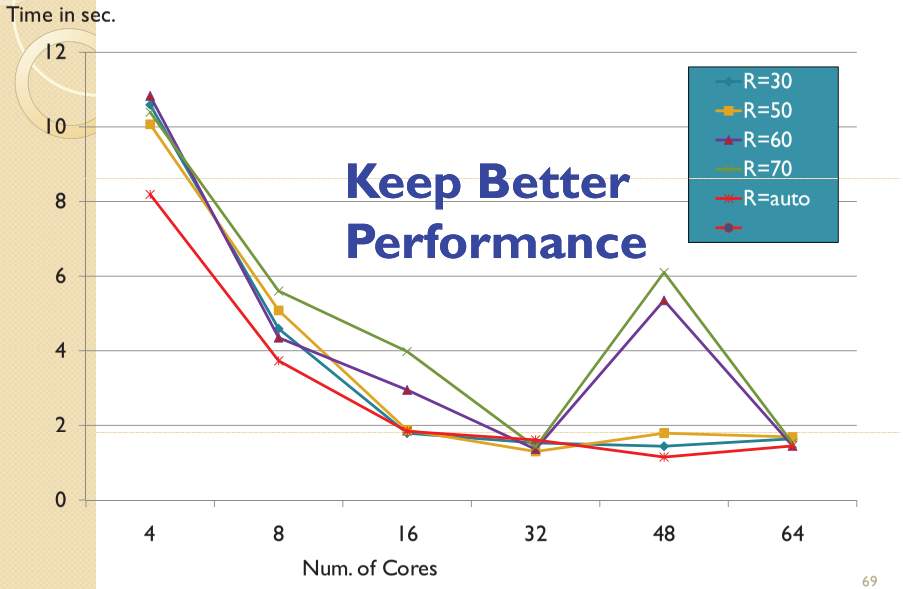
48 cores (3 nodes)

Iter.	Orth.	Time
5039	0.59	1.44
-	-	-
5580	0.89	1.79
3715	4.89	5.35
3630	5.55	6.10
4237	0.52	1.15

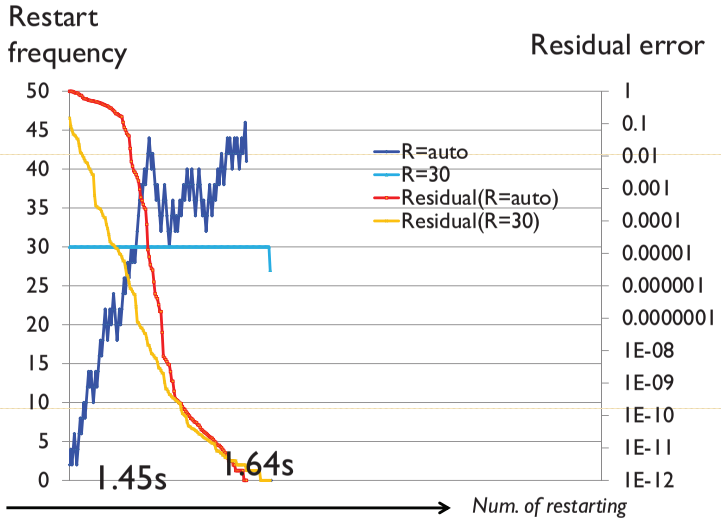
64 cores (4 nodes)

Iter.	Orth.	Time
5127	0.79	1.64
-	-	-
4648	0.84	1.69
4371	0.84	1.45
3323	0.67	1.52
4455	0.68	1.45

chem_master1 on The T2K (Good Case)



chem_master1 on The T2K (64cores)



epb3 on The T2K

4 cores

Restart	Iter.	Orth.	Time
96	36701	249.77	292.24
100	24181	172.83	208.87
120	20479	176.50	207.00
140	18963	192.40	220.44
160	15592	193.59	220.64
Auto	22001	178.59	211.57

8 cores

Iter.	Orth.	Time
29867	126.07	142.20
23060	103.35	120.41
20753	98.75	111.52
15881	95.35	104.49
15548	101.11	112.84
37384	113.31	135.17

16 cores

Iter.	Orth.	Time
36107	98.78	116.97
24499	69.29	81.74
20757	69.46	79.89
21130	82.44	93.59
16552	72.90	81.57
35309	95.79	113.92

32 cores (2 nodes)

Restart	Iter.	Orth.	Time
96	33228	79.45	113.09
100	22988	57.88	81.00
120	26740	82.96	117.51
140	26958	100.96	142.96
160	16460	71.99	101.02
Auto	35718	77.94	110.48

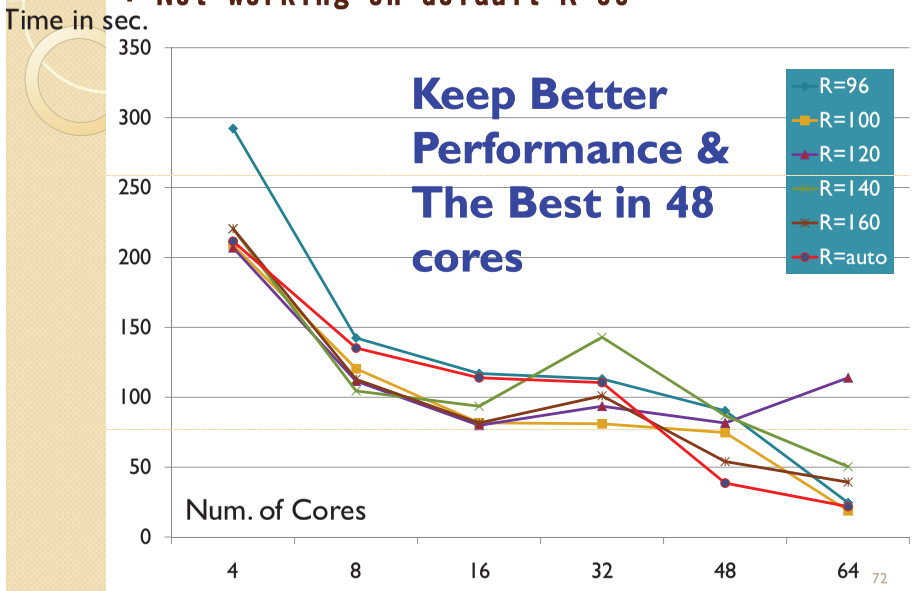
48 cores (3 nodes)

Iter.	Orth.	Time
33492	83.94	90.31
25885	69.35	74.73
23738	77.64	82.57
22029	81.50	86.89
14857	37.54	53.96
34736	27.34	38.58

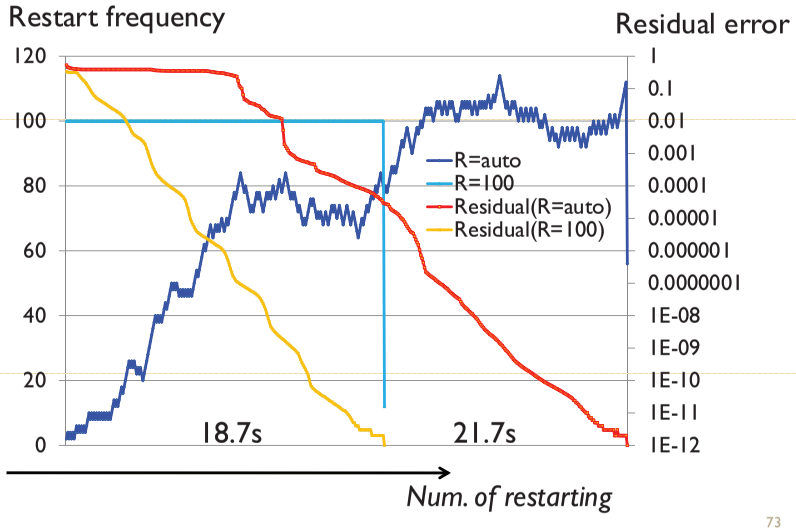
64 cores (4 nodes)

Iter.	Orth.	Time
33790	17.02	24.52
25412	14.07	18.77
28162	26.46	37.03
25529	35.19	50.30
15800	27.62	39.21
32804	14.74	21.77

epb3 on The T2K (Not Bad Case)
: Not working on default R=30



epb3 on The T2K (64cores)



matrix_9 on The T2K

4 cores

Restart	Iter.	Orth.	Time
90	-	-	-
100	-	-	-
120	3105	33.70	129.33
140	1600	19.63	70.94
160	2061	29.06	94.98
Auto	8748	76.59	353.55

8 cores

Iter.	Orth.	Time
-	-	-
2960	17.24	62.70
1799	12.43	40.87
1857	13.83	42.97
1871	16.52	45.22
8504	35.95	175.67

16 cores

Iter.	Orth.	Time
2803	13.90	35.98
2227	12.53	29.98
1956	10.30	27.18
1605	9.44	22.48
1430	9.35	20.68
7468	34.69	98.47

32 cores (2 nodes)

Restart	Iter.	Orth.	Time
90	2643	12.53	22.10
100	2814	15.86	26.49
120	1751	10.56	17.45
140	1662	10.92	17.54
160	1461	10.29	21.19
Auto	8282	77.94	63.72

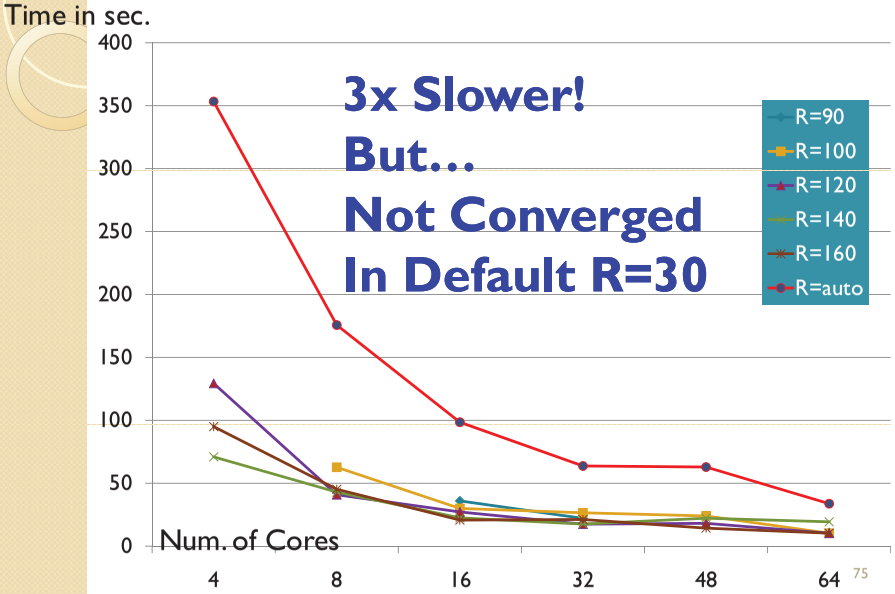
48 cores (3 nodes)

Iter.	Orth.	Time
-	-	-
3435	15.30	23.97
2583	12.00	18.14
1876	9.88	22.17
1729	9.78	14.32
12449	31.46	62.85

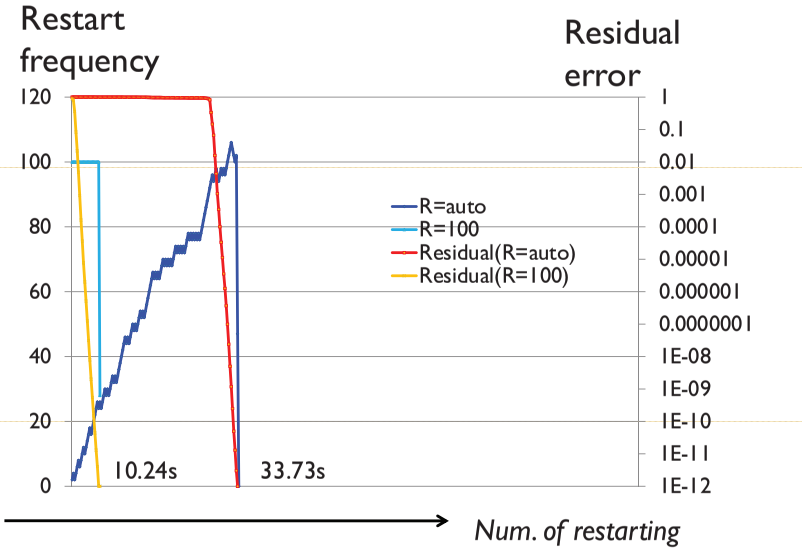
64 cores (4 nodes)

Iter.	Orth.	Time
2388	5.99	10.80
2228	5.31	10.24
1795	5.69	10.24
1666	4.71	19.31
1587	5.50	10.30
7587	14.74	33.73

matrix_9 on The T2K (Bad Case) : Not working on default R=30



matrix_9 on The T2K (64cores)

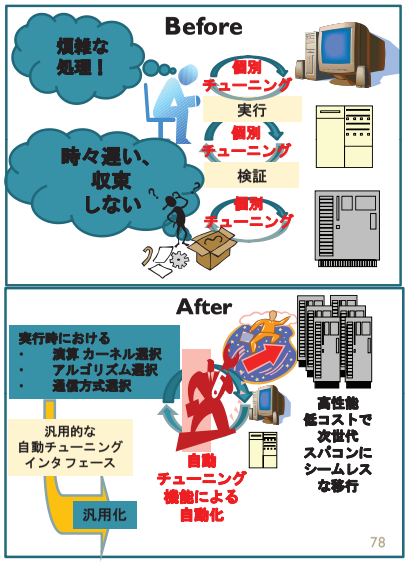


反復解法ソルバ XABCLIB (OPENMP並列化版)

※愛媛大学　黒田久泰 先生
東京大学　中島研吾 先生
日立製作所　直野健 博士
日立製作所　櫻井隆雄 氏
との共同研究

e-サイエンス実現のためのシステム統合・連携ソフトウェアの研究開発
高生産・高性能計算機環境実現のためのシステムソフトウェアの研究開発
「シームレス高生産・高性能プログラミング環境」高性能高可搬性ライブラリ
（平成20年度～平成23年度）：代表　東京大学　石川裕　教授

- 問題
  - 最適化が職人芸に依存 、生産性がない、可搬性がない、 煩雑な処理、コストがかかる
  - 理論的に範囲外のパラメタ指定による収束失敗
- 目標
  - 高生産性／性能移植性をもつ数値計算ライブラリの提供
- 疎行列の非ゼロ成分の分布を考慮し
  実行時における以下の自動チューニング機能を提供
  1. ***演算カーネル選択***
  2. ***数値アルゴリズム選択***
  3. ***並列実装方式選択***
  4. ***汎用的なAPI***



E-Science Development Library

- Development of 2009 (Development From Scratch)
  - **Xabclib_LANCZOS**
    - Eigensolver with the Restart LANCZOS for Standard Eigenproblem
  - **Xabclib_GMRES**
    - Linear Equations Iterative Solver with GMRES(m)
  - **OpenATlib**
    - **Common AT Interface Library**
- The following AT Facilities are implemented:
  1. General Auto-tuning Facility for Re-start Frequency
  2. Run-time Selection for SpMxV Implementations
  3. Optimization for Multicore Processors
     - Optimized for invocated number of processes
  4. **Run-time Implementation Selection Based on Run-time Memory Restriction**
  5. Common AT Interface to Establish The above AT Functions
- Fortran and OpenMP Parallelization, CRS format

OpenATlib: A General AT Interface for Library Developers

- **Restart Frequency Interface**
  OpenATI_DAFRT(NSAMP, SAMP, IRT,INFO)
  - NSAMP: The Number of Sampling Points
  - SAMP: Sampling Data (double)
  - IRT: Judgment Flag
    - 0 : Do not need Increase
    - 1 : Need Increase
  - **Judgment Function : MM ratio [T.Sakurai,2008]**

$$R_i(s,t) = \max_z \{ r_i(z); z=s-t+1,\ldots,s\} / \min_z \{r_i(z); z=s-t+1,\ldots,s\}$$

Is a ratio for the max per min on residuals r_i from s-t+1 to s.

Extension to ILIB: Last S samples.

An Example of OpenATI_DAFRT

```
INCLUDE "OpenAT.inc"
MSIZE=1 //First Restart Frequency
I=5 //Judgment Frequency
    ...
IF RSDID < TOL RETURN //Convergence Test
SAMP (K)=RSDID // Set Residual to SAMP(K)
IF K = I THEN //Call DAFRT per I-times
    IRT=0
    CALL OpenATI_DAFRT (I, SAMP, IRT, INFO)
    IF IRT= 1 MSIZE=MSIZE+1 //Increase Restart Frequency
    K=0
END IF
K=K+1
```


OpenATIib: A General AT Interface for Library Developers

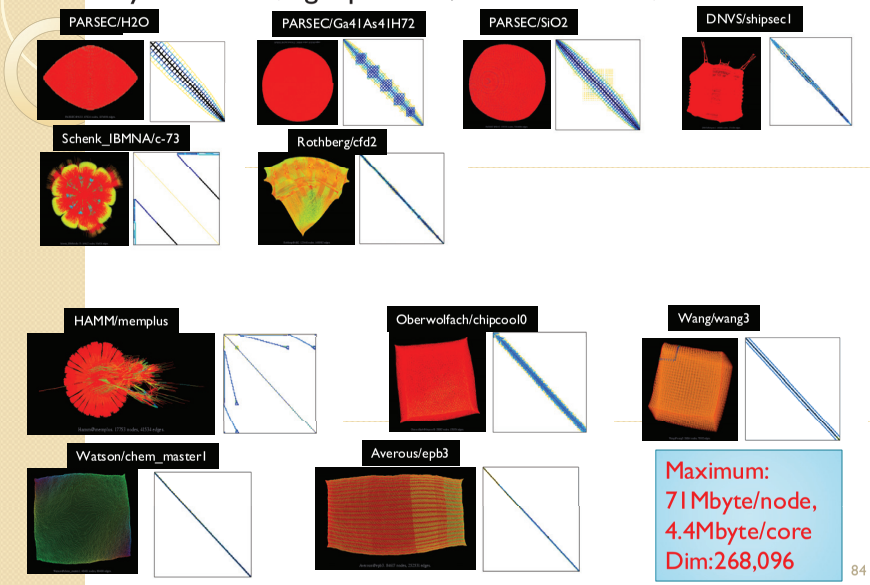
- **SpMxV Interface (y = Ax)**
  OpenATI_DSRMV() : Symmetric
  OepnATI_DURMV() : Unsymmetric
  - Arguments:
    - N : Matrix Size
    - NNZ : Number of Nonzero Elements
    - IRP(N+1) : Diagonal Index Pointer
    - ICOL(NNZ) : Row Index Pointer
    - VAL(NNZ) : Element Value
    - X(N) : RHS Vector
    - Y(N) : LHS vector
    - **ICASE : AT Implement Switch No**
    - **NUM_SMP: Reduction Number**
    - **WK(N, NUM_SMP) : Work Space**
    - **INFO : Error Code**

An Example of OpenATI_DSRMV

```
INCLUDE "OpenAT.inc"
OpenATI_DSRMV_IPARM_1=3  //Set DSRMV Sampling Mode
ICASE=0  // Clear DSRMV Parameter
  …
//First SpMxV
CALL OpenATI_ DSRMV (N, NNZ, IRP, ICOL, VAL, X, Y,
          ICASE, NUM_SMP, WK, INFO)
OpenATI_DSRMV_IPARM_1=1
    //After them, the selected method is used.
    …
//The set parameter is used.
CALL OpenATI_ DSRMV (N, NNZ, IRP, ICOL, VAL, VEC,
JPARM, IPARM, RPARM, INFO)
    …
```


U. Florida Sparse Matrix Collection

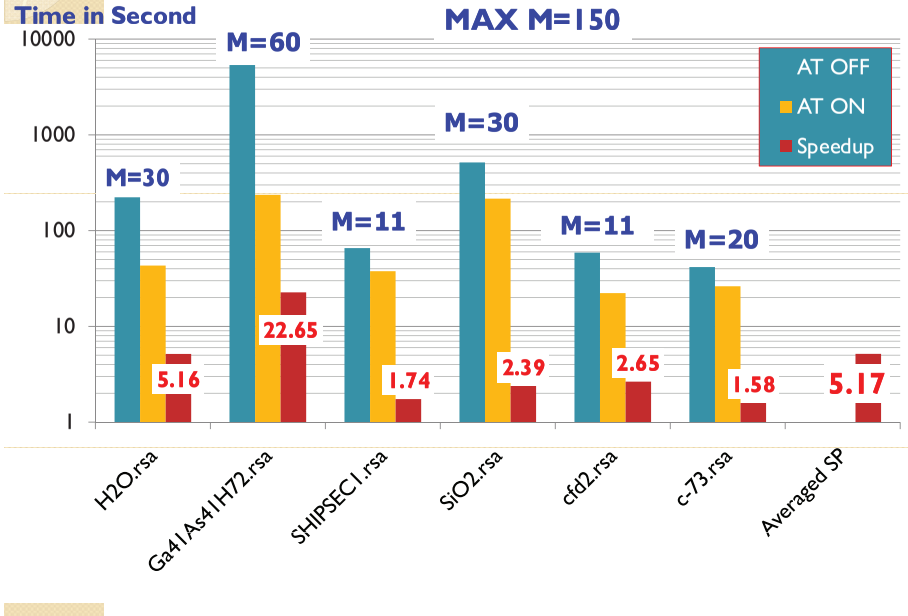
- Symmetric (Eigenproblem, Lancoz Method)



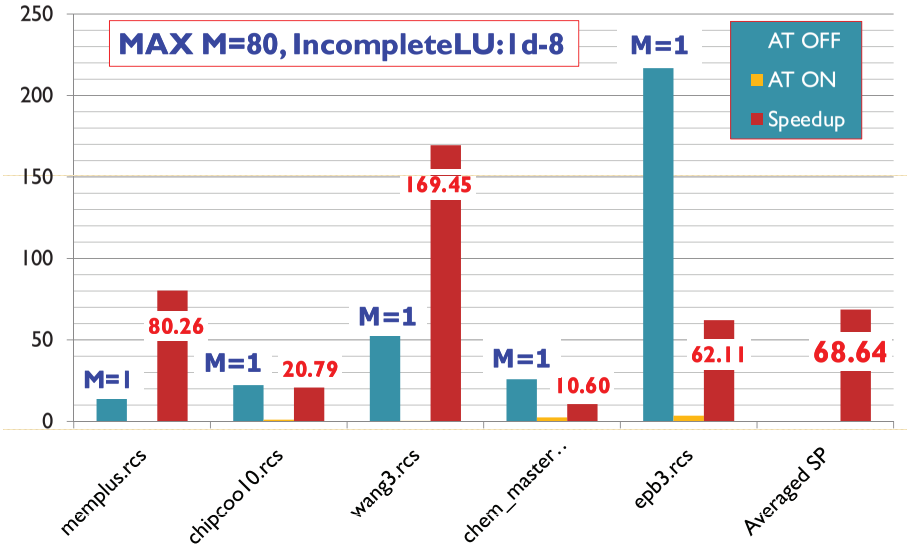
Evaluation Environment

- T2K Open Supercomputer(Todai)
- One Node Execution with OpenMP Parallelization (Maximum 16 cores on one node)
- Compiler
  - Intel Fortran Compiler Professional Version 11.0
- Compiler Option:
  - -O3 -m64 -openmp -mcmodel=medium
- **AT Facility**
  - Restart Frequency
  - SpMxV implementation selection
    - **a) Normal; b) 8-2 unroll; c) Vectorized;**
    - **Implementation Selection According to Memory Usage**
- Algorithm Specialized Matters
  - LANCZOS: 10 eigenvalues computations
  - GMRES : The Jacobi Preconditioner
    - Other Choices: Scaling, Jacobi, ILU, SSOR

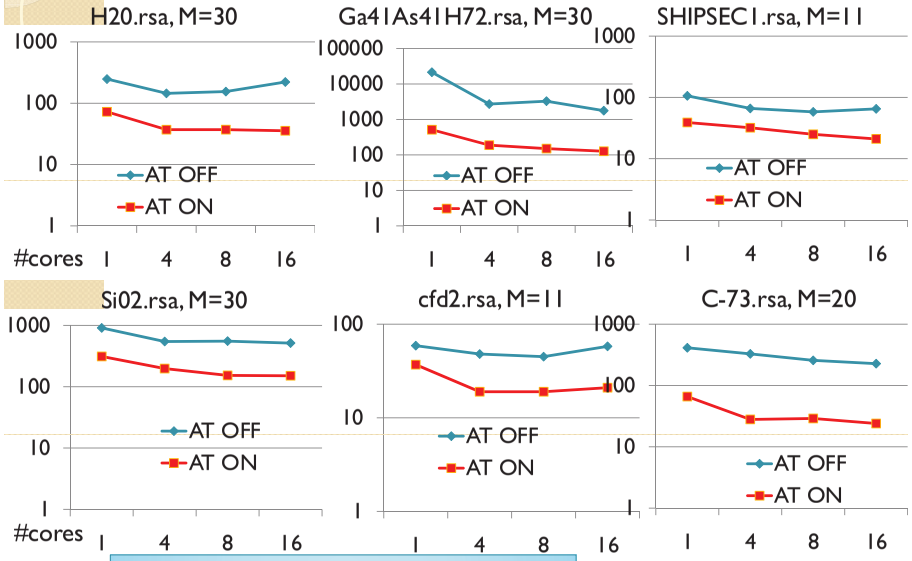
Restart Frequency AT Result (LANCZOS, 16 cores)



Restart Frequency AT Result (GMRES, 16 cores)

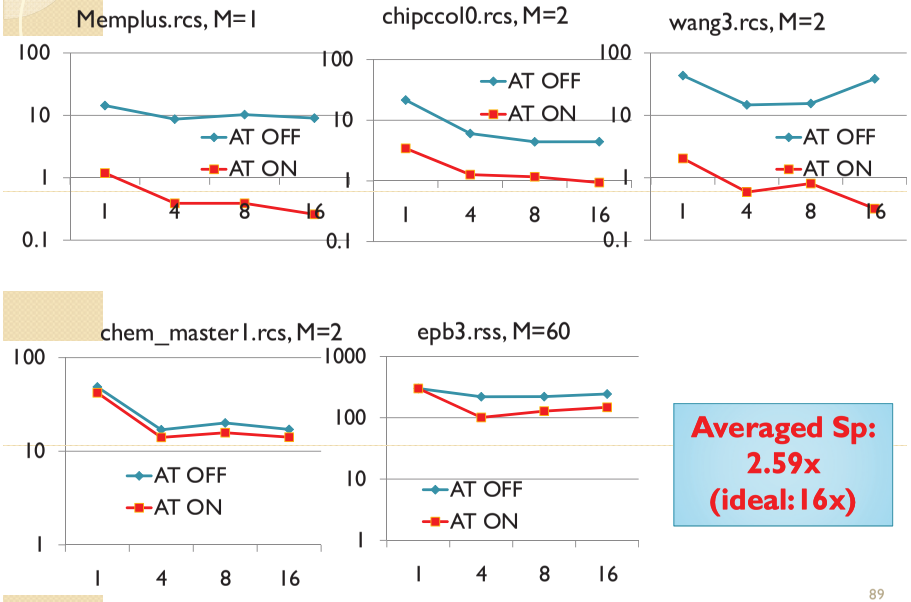


Parallel Performance (LANCZOS)



Averaged Sp:1.98x (ideal:16x)

Parallel Performance (GMRES, Jacobi Preconditioner)



Averaged Sp: 2.59x (ideal:16x)

残された話題

- 数値アルゴリズム関連（以下は事例の一部）
  - 自動チューニング機構
    1. 解法全般に通用する性能パラメタの自動チューニング手法
    2. ユーザが望む数値計算ポリシー（速度／精度／メモリ量など）を考慮した自動チューニング手法
       - ＜精度保障技術＞の導入も考慮
    3. 選択（推定）パラメタと性能モデルの精度解析と自動補正
  - 数値解法
    1. 反復解法における各種性能パラメタの自動チューニング
       - 前処理方式選択
       - 再直交化など、解法に依存した内部アルゴリズム選択
    2. 疎行列直接解法におけるオーダリング方式自動選択
    3. 疎行列データ形式の自動選択
       - 各種の疎行列圧縮形式（と、データ再変換）
       - 行列 - ベクトル積の実装方式（通信方式、負荷バランス調整）
- 計算機システムの問題も山積

おわりに

- ソフトウェア自動チューニング研究は＜学際領域＞
  - 計算科学とコンピュータサイエンスの境界
    1. 線形計算／最適化の数理
    2. 先進計算機アーキテクチャ／システム
    3. コンパイラ／言語処理
    4. 数値計算ライブラリ
    5. 数値計算アルゴリズム
    6. 超並列アルゴリズム
    7. ソフトウェア工学／品質管理

    の全般の知識を駆使しないと解決できない。
- 実学、今後5年以内に必須となる分野
- 学生／若手研究者の活発な参入に期待

自動チューニング研究会

- 2003年　電通大で発足
  - 初期メンバー：
    弓場、直野、今村、須田、山本、片桐
- 自動チューニング研究の情報交換、研究資金／研究企画の調整、国際会議企画(iWAPT)、等を行う団体
- 現在、会員18名
- 入会したい方は片桐までご連絡を
- 参考資料
  - 第4回iWAPT2009ホームページ（http://www.iwapt.org/）
    - 2009年10月1日、2日、東京大学小柴ホール
    - Paper submission due: June 29th, 2009
    - 参加費無料
    - ふるってご投稿＆ご参加を！

謝辞

- 自動チューニング研究会（順不同）
  1. 主査：須田礼仁（東大）
  2. 幹事（総務）：片桐孝洋（東大）
  3. 幹事（連絡）：伊藤祥司（理研）
  4. 顧問：弓場敏嗣（電通大名誉教授）
  5. 今村俊幸（電通大）
  6. 山本有作（名大）
  7. 直野健（（株）日立製作所中央研究所）
  8. 清水謙多郎（東大）.
  9. 佐藤周行（東大）
  10. 岩下武史（京大.）
  11. 寺内和也（日本ビジュアルニューメリックス）
  12. 恵木正史（（株）日立製作所中央研究所）
  13. 櫻井隆雄（（株）日立製作所中央研究所）
  14. 黒田久泰（愛媛大）
  15. 中島研吾（東大）
  16. 滝沢寛之（東北大）
  17. 高橋大介（筑波大）
  18. 八杉昌宏（京大）

参考文献

第一部

1. 片桐孝洋：「ソフトウエア自動チューニング－数値計算ソフトウエアへの適用とその可能性」、慧文社、2004年12月3日初版第一刷発行、ISBN4-905849-18-7
2. 情報処理学会学会雑誌「情報処理」2009年6月号特集「科学技術計算におけるソフトウェア自動チューニング」

第三部（参考資料）

1. 片桐孝洋，吉瀬謙二，本多弘樹，弓場敏嗣，２００４年先進的計算基盤システムシンポジウム（Symposium on Advanced Computing Systems and Infrastructures (SACSIS)，２００４年５月２６日（水）～２８日（金）、札幌コンベンションセンター，ＳＡＣＳＩＳ２００４論文集，pp. 43--52 (2004)：「自動チューニング処理記述用ディレクティブABCLibScriptの設計と実装」
2. Takahiro Katagiri, Kenji Kise, Hiroki Honda, and Toshitsugu Yuba, Parallel Computing, Vol.32, Issue 1, pp.92-112 (January 2006): "ABCLibScript: A Directive to Support Specification of An Auto-tuning Facility for Numerical Software"

参考文献

第二部・第四部（参考資料）

1. Takahiro Katagiri, Kenji Kise, Hiroki Honda, and Toshitsugu Yuba, Parallel Computing, Vol.32, Issue 3, pp.231-250 (March 2006) "ABCLib_DRSSED: A Parallel Eigensolver with an Auto-tuning Facility"
2. Takahiro Katagiri, Christof Voemel, James W. Demmel, 第13回「ハイパフォーマンスコンピューティングとアーキテクチャの評価」に関する北海道ワークショップ（HOKKE-2006）、北海道大学 学術交流会館 第一会議室、情報処理学会研究報告２００６－ＨＰＣ－１０５、pp.25--30、2006年2月27日（月） (2006) "Multi-section with Multiple Eigenvalues Method for Computing Eigenvalues in Symmetric Tridiagonal Eigensolvers"
3. Takahiro Katagiri, Christof Voemel, James W. Demmel, ２００６年並列／分散／協調処理に関する『高知』サマー・ワークショップ（ＳＷＯＰＰ2006）、高知商工会館、情報処理学会研究報告２００６－ＨＰＣ－１０７、pp.187--192、2006年7月31日（月）--8月2日（水） (2006) "Effect on Run-time Auto-tuning for Multi-section with Multiple Eigenvalues Method"
4. 片桐孝洋、情報処理学会研究報告２００８－ＨＰＣ－１１７、pp.31--36、2008年10月15日（水）、汐留シティセンター、(2008)「超並列マルチコア環境での自動チューニング機能の有効性：T2Kオープンスパコン上の固有値ソルバを例にして」
5. 片桐孝洋、黒田久泰、２００８年並列／分散／協調処理に関する『旭川』サマー・ワークショップ（ＳＷＯＰＰ2008）、佐賀、アバンセ、情報処理学会研究報告２００８－ＨＰＣ－１１６、pp.43--48、2008年8月5日（火）--8月7日（木） (2008)「Windowsクラスタにおける疎行列反復解法ソルバの自動チューニング」
6. 片桐孝洋、東京大学情報基盤センター スーパーコンピューティングニュース、Vol.11, No.1, pp. 47-59 (2009.1)「ペタスケール計算機環境に向けた固有値ソルバの開発」
7. 片桐孝洋、黒田久泰、第14回日本計算工学会講演会、オーガナイズドセッション、「科学技術計算における自動チューニングの挑戦：マルチコア環境への適用」、東京大学生産研究所、2009年5月12日（火）－5月14日（木）、計算工学会講演論文集、pp.167--170 (2009)「マルチコア環境における自動チューニング機能付き疎行列反復解法ライブラリ」
8. 片桐孝洋、第14回日本計算工学会講演会、オーガナイズドセッション、「解けたらうれしい固有値問題」、東京大学生産研究所、2009年5月12日（火）－5月14日（木）、計算工学会講演論文集、pp.185--188 (2009)「ペタスケール計算を目指したMRRR法を用いた固有値ソルバの開発」

参考資料

講演内容

- 第一部：ソフトウェア自動チューニング概論
  - 数理からみた自動チューニングとは
  - 計算機システムからみた自動チューニングとは
- 第二部：自動チューニング機能付き数値計算ライブラリ
  - 密行列固有値ソルバ関連（Householder三重対角化、QR分解）
  - 疎行列連立一次方程式ソルバ（GMRES法）
- 第三部：自動チューニング記述用計算機言語
  - ABCLibScript（日本発／初のＡＴ言語、ＡＴ研究成果物の一つ、ある意味世界の先駆け）
  - 行列積計算に対するABCLibScriptのデモンストレーション
- 第四部：超並列固有値解析アルゴリズム
  - 並列固有値ソルバ（逆反復法、MRRR法、dqds法）

第三部

自動チューニング機能を記述するための専用言語

- チューニングの「手間」を削減する
- チューニングの「つまみ」を自動生成する

言語設計方針

- 容易に自動チューニング指定
  - 数値計算処理に機能限定:
    - アンローリング指定子(*unroll*)
    - 変動パラメタ指定子(*variable*)
    - アルゴリズム選択指定子(*select*)
- もとのプログラムの実行を阻害しない
  - ディレクティブ形式でプログラム中に記述
- 高い可読性コードを自動生成
  - Fortran90＋MPI-1のコードを自動生成
  - 仕様は計算機言語には依存しない
- 自動チューニングの見通しがよい
  - ２種パラメタ（ＢＰ、ＰＰ）概念の導入

プリプロセッサの処理

ディレクティブで記述

```
!ABCLib$ install unroll (i) region start
!ABCLib$ name MyMatMul
!ABCLib$ varied (i) from 1 to 8
do i=1,N
  do j=1,N
    da1 = A(i,j)
    do k=1,N
      dc = C(k,j)
      da1 = da1 + B(i,k) * dc   enddo
    A(i,j) = da1
enddo   enddo
!ABCLib$ install unroll (i) region end
```

自動生成

パラメタ最適化コンポーネント
- ■最適化
- ■実測とモデル化

AT領域選択コンポーネント
- ■実行時最適化
- ■最適化済みパラメタ設定

AT領域ライブラリコンポーネント
- ■チューニング対象領域のサブルーチン／ライブラリ化

インストール時自動チューニング指定：行列積コードのループアンローリング

```
!ABCLib$ install unroll (i) region start
!ABCLib$ name MyMatMul
!ABCLib$ varied (i) from 1 to 8
do i=1, N
   do j=1, N
      dal = A(i, j)
      do k=1, N
         dc = C(k, j)
         dal = dal + B(i, k) * dc
      enddo
      A(i, j) = dal
   enddo
enddo
!ABCLib$ install unroll (i) region end
```

インストール時自動チューニング
アンロール処理の指定

最大アンローリング段数指定
（ソフトウエア開発者の知識に基づくマジックナンバー）

対象領域
（ＡＴ領域）
（ソフトウエア開発者が知っている）

行列積コードのループアンローリング（続き）

- プリプロセッサ実行後、最外側の *i* ループがアンロールされ、AT領域ライブラリコンポーネントに自動登録

```
if (i_unroll .eq. 1) then
   Original Code
endif
if (i_unroll .eq. 2) then      /* i is dividable by 2 */
   im = N/2
   i = 1
   do ii=1, im
      do j=1, N
         dal = A(i, j); da2 = A(i+1,j)
         do k=1, N
            dc = C(k, j)
            dal = dal + B(i,  k) * dc;  da2 = da2 + B(i+1, k) * dc; enddo
         A(i, j) = dal;A(i+1,j) = da2
      enddo
      i = i + 2;
   enddo
endif
```

アンローリング段数が
自動的にパラメタ化される

インストール時自動チューニング指定：ブロック幅調整

```
!ABCLib$ install variable (MB) region start
!ABCLib$ name BlkMatMal
!ABCLib$ varied (MB) from 1 to 64
do i=1, n, MB
  call MyBlkMatVec(A,B,C,n,i)
enddo

!ABCLib$ install variable (MB) region end
```

変動パラメタの
自動チューニング指定

ブロック幅調整指定
（１から６４まで）

対象領域
（ＡＴ領域）
（ソフトウエア開発者が知っている）

実行起動前自動チューニング指定：アルゴリズム選択

実行起動前自動チューニング指定、
アルゴリズム選択処理の指定

```
!ABCLib$  static  select region start
!ABCLib$  parameter (in CacheS, in NB, in NPrc)
!ABCLib$       select sub region start
!ABCLib$       according estimated
!ABCLib$          (2.0d0*CacheS*NB)/(3.0d0*NPrc)
                  対象１（アルゴリズム１）
!ABCLIB$       select sub region end
!ABCLib$       select sub region start
!ABCLib$       according estimated
!ABCLib$          (4.0d0*CbcheS*dlog(NB))/(2.0d0*NPrc)
                  対象２（アルゴリズム２）
!ABCLib$       select sub region end
!ABCLib$ static select  region  end
```

コスト定義関数
の入力変数

コスト定義関数
（ＡＴ領域選択基準）
（ソフトウエア開発者が知る知識）

対象領域１、２
(AT領域１、２)
（ソフトウエア開発者が知る知識）

対象１と２の選択情報が
パラメタ化される

ABCLibScriptのデモ

- 自動チューニングに必要なコードを、いまここで自動生成してみる
  - 「行列-行列積」コード
    - ちょっと普通でない書き方
      - 最適化されているコードなど、実用コードは素直でない
  - ループアンローリングの適用事例
  - i, j, k 三重ループにおける、それぞれ8段展開
    - 自動生成のコードの種類：
      8×8×8 =512種類
    - 手書きで開発するのは事実上無理
    - コード自動生成なら、製作は一瞬
      - でも実行（チューニング）はそれなりに時間がかかる
      - だが、コード製作作業、チューニングのための測定作業が完全に自動化されることは大きい

講演内容

- 第一部：ソフトウェア自動チューニング概論
  - 数理からみた自動チューニングとは
  - 計算機システムからみた自動チューニングとは
- 第二部：自動チューニング機能付き数値計算ライブラリ
  - 密行列固有値ソルバ関連（Householder三重対角化、QR分解）
  - 疎行列連立一次方程式ソルバ（GMRES法）
- 第三部：自動チューニング記述用計算機言語
  - ABCLibScript（日本発／初のＡＴ言語、ＡＴ研究成果物の一つ、ある意味世界の先駆け）
  - 行列積計算に対するABCLibScriptのデモンストレーション
- 第四部：超並列固有値解析アルゴリズム
  - 並列固有値ソルバ（逆反復法、MRRR法、dqds法）

第四部

超並列数値計算アルゴリズム

●先進アーキテクチャにマッチしたチューニングの「つまみ」を開発する

ペタフロップスマシンの登場

- 2008年11月のTOP500
  - 1位：Roadrunner (DOE/NNSA/LANL)
    1.105 PFLOPS、129,600 コア
  - 2位：Jaguar (ORNL)
    1.059 PFLOPS、150,152 コア
- ペタフロップスマシンのコア数は１０万コア以上
  - 大規模数値シミュレーションは、１アプリで１０万並列が必要（期待される）
  - ソルバ部分（数値計算ライブラリ）が大部分の実行時間を占める場合が多い
- ソルバは１０万並列を達成できるアルゴリズム／実装になっていますか？

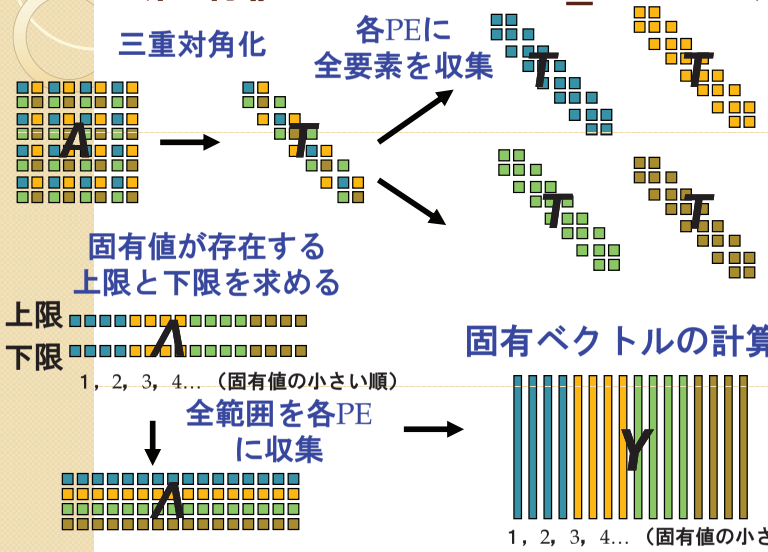
話の流れ

- 背景
- 対称実数密行列固有値ソルバ
  - アルゴリズムの説明
    - Householder三重対角化を経由するアルゴリズム
    - 先進的固有値アルゴリズム：MRRR
  - 性能評価
    - 性能評価環境：T2Kオープンスパコン（東大版）
    - 単体性能評価
    - ピュアMPI実行 vs ハイブリッド実行
    - 大規模問題実行性能

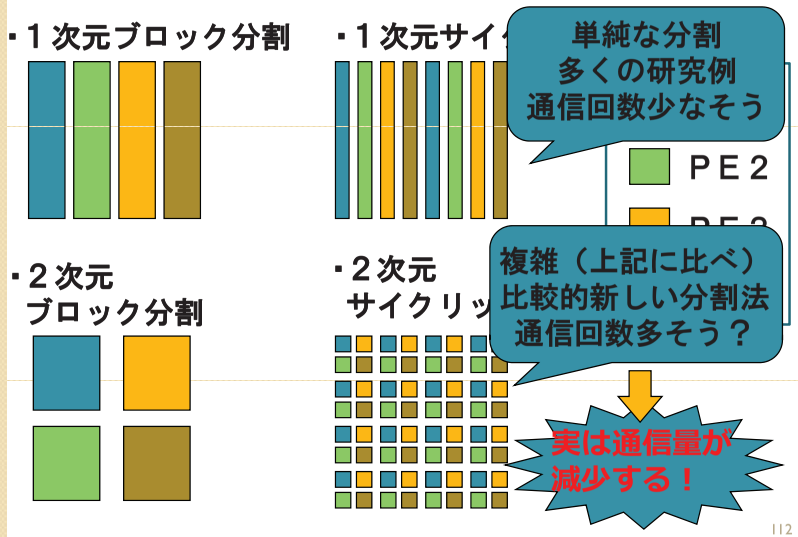
用途に応じた最適化戦略

- 前提：全て（もしくは次元数の半数以上）の固有値・固有ベクトルを求める場合
- 大規模固有値計算を１回（"普通"の大規模計算）
  - 三重対角化がネック
  - キャッシュブロック化手法を駆使し、単体性能を向上させる手法がよい
- 小～中規模固有値問題を数百回～数千回（計算時間が大規模）
  - 三重対角化がネック
  - 場合により固有ベクトル計算部分がネック
  - 単体性能より台数効果を向上させる手法がよい
    - 固有値ソルバよりその他の処理（行列生成など）が重い
    - 上記理由から
      - 全体実行時間制約により、固有値ソルバ部分の次元数が大規模になり得ない（大規模な場合の1/5~1/10）
      - 配列データが完全分散の場合が多い →プロセス数を少なくして使うのが面倒／性能出ない

並列ソルバ全体の流れ（固有値ソルバABCLib_DRSSED）

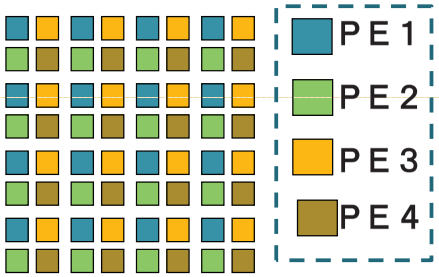


データ分散方式の分類



Householder三重対角化の通信削減アルゴリズム

2次元サイクリック分割方式



＋ 通信削減方式

欠点 •通信回数が（他の分割に比べ）多い

利点 •負荷バランスが完全

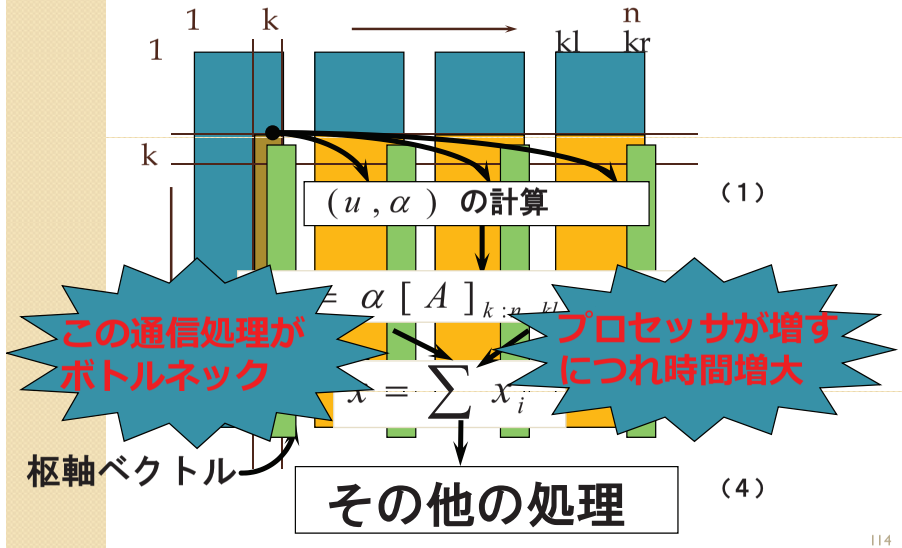
•*通信量が削減可能*

p ：プロセッサ台数

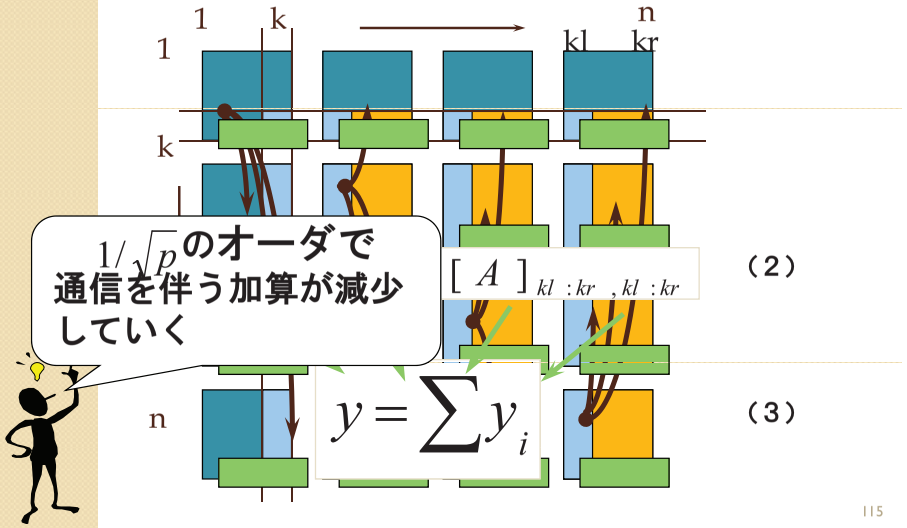
n ：問題サイズ

$$O(n^2 \log_2 p) \longrightarrow O(n^2 / \sqrt{p} \cdot \log_2 \sqrt{p})$$

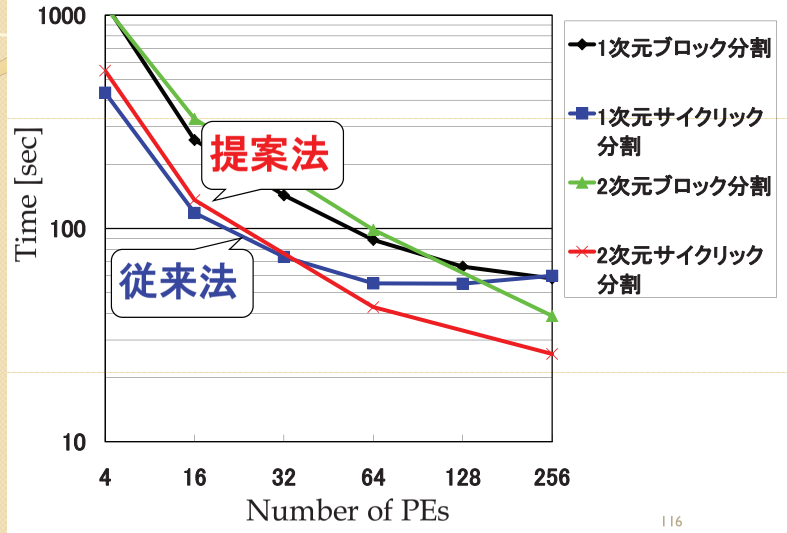
１次元サイクリック分割（*従来法*）の並列処理の流れ



２次元サイクリック分割方式（提案法）の並列処理の流れ



日立SR2201@東大大計算センタ（2000年）の実行結果（Hessenberg化： $n = 4096$）



話の流れ

- 背景
- 対称実数密行列固有値ソルバ
  - アルゴリズムの説明
    - Householder三重対角化を経由するアルゴリズム
    - 先進的固有値アルゴリズム：MRRR
  - 性能評価
    - 性能評価環境：T2Kオープンスパコン（東大版）
    - 単体性能評価
    - ピュアMPI実行 vs ハイブリッド実行
    - 大規模問題実行性能

逆反復法

•得られている $\hat{\lambda}$ と、適切な v_0 を用いて、以下を反復：

•逆反復：$\tilde{v}_{k+1} = (T - \hat{\lambda} I)^{-1} \tilde{v}_k$

•正規化：$\tilde{v}_{k+1} = \tilde{v}_{k+1} / \|\tilde{v}_{k+1}\|$

• $\hat{\lambda}$ が密集している場合

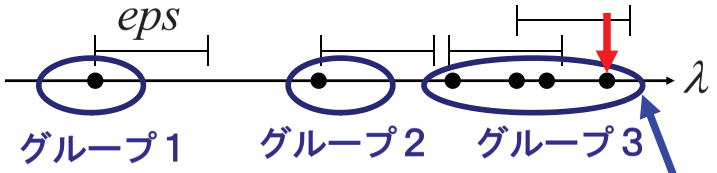
•Gram-Schmidt法を用いた直交化

•これより前に計算した密集固有値に対する固有ベクトルに 対し $\tilde{v}_{k+1}$ を直交化

•収束判定

密集固有値の判定：グループ化

- 固有ベクトルを直交化する範囲
  - 固有ベクトルに対応する固有値が指定距離 eps よりも近い範囲内



Peters-Wilkinson のグループ分け方式

さらに eps の距離に固有値を含むとき，その固有値も同グループとする

並列逆反復法アルゴリズム [片桐ら,98]

do iter=1, maxiter
- If (グループ内番号１の固有ベクトルを計算中) then
  - PE内でグループ内番号１の固有ベクトルを計算 endif
- If (計算中の固有ベクトルがグループ内番号１番の固有ベクトルでない) then
  - If (計算に必要な固有ベクトルが全部自分のPE内にある) then
    - 固有ベクトルの計算 endif
  - If (計算に必要な固有ベクトルが全部PE内にない) then
    - （グループ番号の若い順に） グループ内番号１を所有するプロセッサに，現在計算中の固有ベクトルを転送
    - If (修正G-S法) (自PE番号-１) のPEから受信待機； 計算；
    - If (古典G-S法) グループ内番号１を所有するプロセッサ～(自PE番号-１) まで受信待機； 計算； endif

  endif
  - 収束判定； ループ強制終了；

enddo
- If (PE内未計算固有ベクトルがない) then
  - If (PEにまたがったグループがある) then
    - 直交化処理のための受信待ち endif endif

全固有値が同一グループ：直交化以外の並列性がない！

従来法の問題点

- 固有値が密集（絶対値の差が小さい）
  - 固有ベクトル計算の際、逆反復法中の直交化の計算量が増大
  - 多くの場合、並列化できない
  - プロセッサ台数が増すにつれ、直交化時間の割合が増大
  - 計算時間の９０％以上が直交化時間となる
- 先進的アルゴリズムMRRRでは、直交化処理が不要に！
  - 直交化しない逆反復法の並列性は理論上きわめて高い

先進的アルゴリズムMRRR：概略

- 固有値解法の先進アルゴリズム[1997, Parlett]
- 実数対称三重対角行列用は、LAPACK 3.1の ***xSTEGR*** ルーチンに搭載
- GR は Holy Grail(聖杯)の略
- 特徴：
  - ☺固有ベクトル計算に、従来必須であった直交化処理が不要
  - ☺$O(N^3) \rightarrow O(N^2)$ に計算量が劇的削減
  - ☺特異値分解処理にも適用可能
  - ☹固有値の密集度合いが相対的に大きいときのみ有効（後述）

先進的アルゴリズムMRRR：概略

- エルミート行列Aに対する標準固有値問題

$$Ax = \lambda x$$

- エルミート行列Aに対する特異値分解

$$A = U\Sigma V^*$$

- k 個の固有値と対応する固有ベクトル 計算に対する、計算量とメモリ量：$O(nk)$
- 残差ベクトル精度： $\|Ax - \lambda x\| = O(n\varepsilon)$
- 数値直交性： $O(n\varepsilon)$
- 自明な並列性をもつ

近接する固有値の直交精度

- 目標： $|v^T w| = O(n\varepsilon)$
  - λ, μ が異なる固有値のとき

$$\|(T - \lambda I)v\| = O(n\varepsilon)$$

$$\|(T - \mu I)w\| = O(n\varepsilon)$$

- ところが、一般的には：

$$O\left(\frac{n\,\varepsilon\,(\|T\|)}{|\mu - \lambda|}\right)$$

三重対角行列Tのノルムが入り目標が達成できない

相対差がある時の固有ベクトル精度

λ：真の固有値　v：真の固有ベクトル
$\hat{\lambda}$：計算した固有値　$\hat{v}$：計算した固有ベクトル

- ２つの固有値が相対的に大きいとき、以下が成立

$$\left\|(T-\hat{\lambda}I)\hat{v}\right\| = O(n\varepsilon\left|\hat{\lambda}\right|)$$

- 計算する固有ベクトルの直交精度は２つの固有値間に大きな相対的ギャップがあれば、以下の小さな相対角度（目標）が達成：

$$\left|\sin\angle(v,\hat{v})\right| \le \frac{\left\|(T-\hat{\lambda}I)\hat{v}\right\|}{gap(\hat{\lambda})} = \frac{O(n\varepsilon\left|\hat{\lambda}\right|)}{gap(\hat{\lambda})}$$

- ただし、$gap(\hat{\lambda}) = \left|\mu-\hat{\lambda}\right|$ で、μ は $\hat{\lambda}$ に最も近い真の固有値である。

２つの固有値の相対差が小さいと目標を達成

MRRR法の特徴

- 以下の新技術の集大成：
  1. T の代わりに LDL^T を使う Relatively Robust Representation (RRR)
  2. Differential qd法 (dqds法)
  3. FP (Fernando-Parlett)ベクトル
  4. 表現木(Representation Tree)と複数のRRR (Multiple RRR, MRRR)

Relatively Robust Representation (RRR)

- 分解 $T-\sigma I = LDL^T$ をするとき、(L,D) は 固有値サブセット Γ におけるRRR
  1. Γ におけるすべての固有値が、それぞれの固有値において相対間隔が大きい
  2. L と Dの要素における小さな相対変動は、λ における小さな相対変動が生じるのみ
- 問題：
  Γ を高い相対精度で計算しないといけない（相対残差が小さくならない）
- 解答：
  dqds法で固有値を高い相対精度で計算

Differential qd法（dqds法）

- 問題：$LDL^T - \sigma I = L_+D_+L_+^T$ をするとき
  - 混合相対安定性：(L_+,D_+) に対する (L,D) 相対変動が小さくなるようにせよ
  - 演算コスト：$O(n)$
- 解答：σ を、計算する $\hat{\lambda}$ に近くなるように、かつ、以下の $\mathrm{relgap}(\hat{\lambda})$ が大きくなるように選ぶ：

$$\mathrm{relgap}(\hat{\lambda}) = \frac{gap(\hat{\lambda})}{\left|\hat{\lambda}\right|}$$

従来：絶対差
MRRR：相対差
→直交化が必要な間隔が減少

- ただし、$gap(\hat{\lambda}) = \left|\mu-\hat{\lambda}\right|$ で、μ は $\hat{\lambda}$ に最も近い真の固有値

FPベクトルと逆反復法の収束

- $\mathrm{relgap}(\hat{\lambda})$ が大きい
  → FPベクトルが小さい相対残差を保障
- 第 j 固有ベクトルに対するFPベクトル：
  - $v_0 = e_j$ ：真の固有ベクトルの j 番目に大きな要素
    （理論的に計算可能）
- 上記の初期ベクトルを用いると、
  たった１回の逆反復で十分収束
  →FPベクトルはきわめてよい初期ベクトル
- 固有ベクトル v_1 は安定した演算で得られる
  （積と割り算、１回の逆反復処理）
- 相対的間隔のもとに直交性は保たれる

MRRRでは、任意の固有ベクトルが
***直交化なし＆逆反復１回* で計算可能**

MRRR：今までの手法を集積

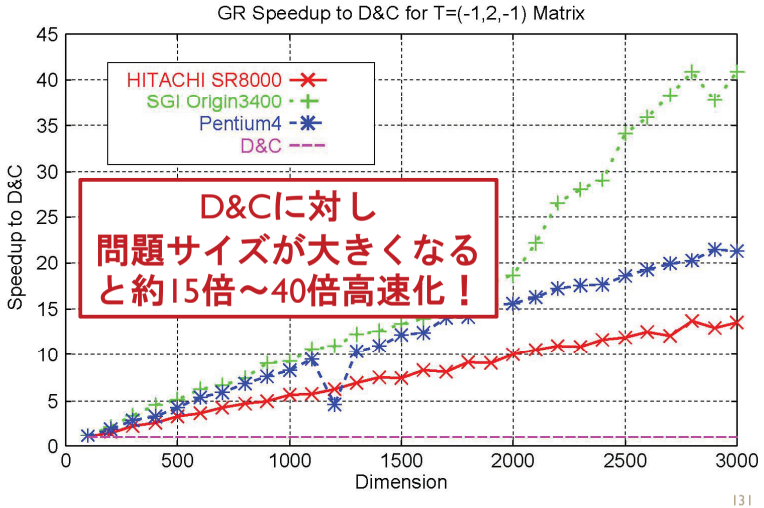
- 固有値集合のRRRが与えられる
  - A) 大きな相対間隔を持つ固有値群
    1. 固有値を相対高精度に求解（dqds法）
    2. FPベクトルの計算（固有ベクトル）
  - B) 残こりの固有値群
    1. その群について、次のシフト値 σ を選ぶ
    2. 新しい RRR を求める（新しい固有値群）

$$L_+ D_+ L_+^T = LDL^T - \sigma I$$

    3. 固有値を改良（二分法）
    4. すべて大きな相対間隔の場合はA) へ
       そうでないなら、B) I.へ

MRRRの典型的性能：速度

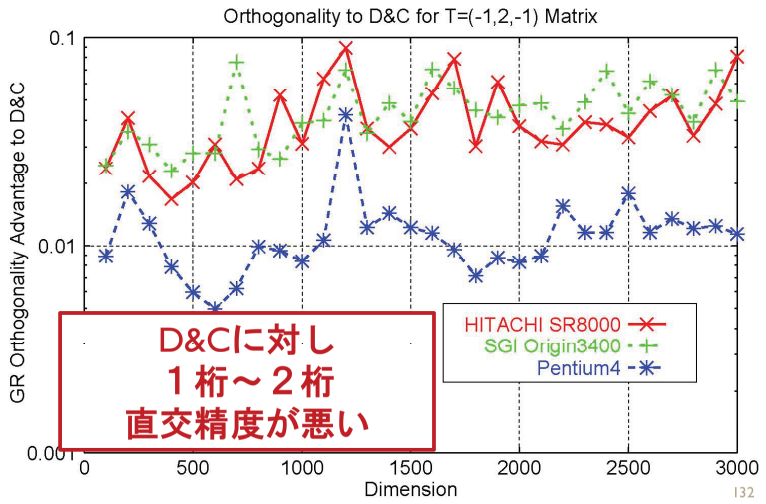
（分割統治法(D&C)に対する速度向上、*T=(-1, 2, -1)*行列）



**D&Cに対し
問題サイズが大きくなると約15倍～40倍高速化！**

MRRRの典型的な性能 ：精度

（分割統治法(D&C)に対する精度低下、*T=(-1, 2, -1)*行列）



**D&Cに対し
１桁～２桁
直交精度が悪い**

話の流れ

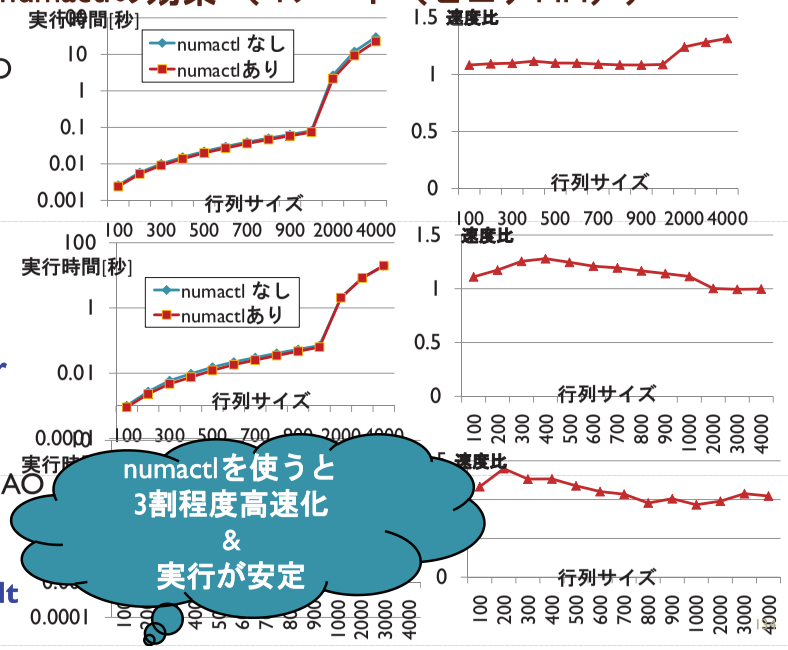
- 背景
- 第１章：対称実数密行列固有値ソルバ
  - アルゴリズムの説明
    - Householder三重対角化を経由するアルゴリズム
    - 先進的固有値アルゴリズム：MRRR
  - 性能評価
    - 性能評価環境：T2Kオープンスパコン（東大版）
    - 単体性能評価
    - ピュアMPI実行 vs ハイブリッド実行
    - 大規模問題実行性能
- 第２章：自動チューニング（AT）機能の開発
  - 適用方式
  - 性能評価
- まとめ

numactlの効果 （１ノード（ピュアMPI））

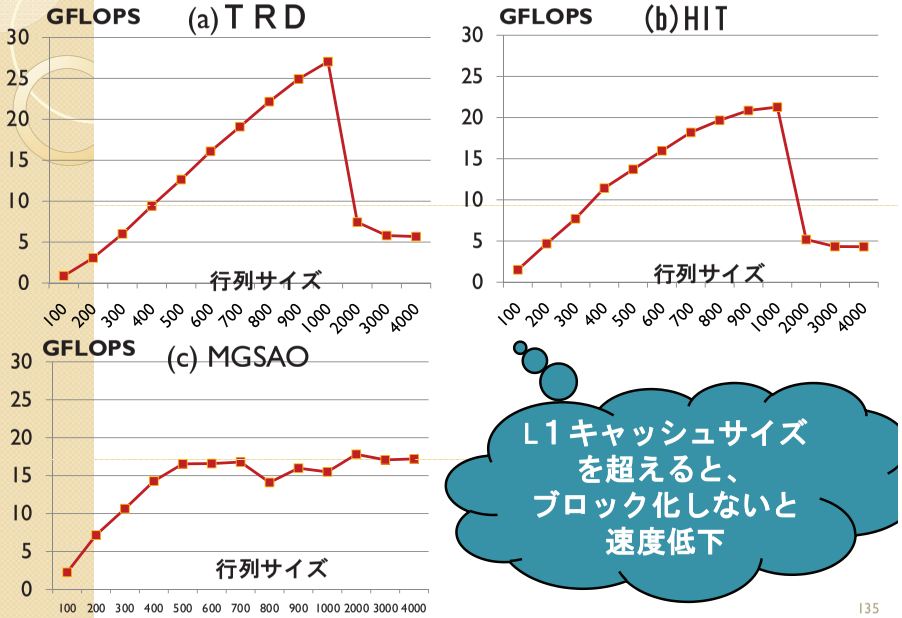
(a) TRD Householder 三重対角化

(b) HIT Householder 逆変換

(c) MGSAO 修正 Gram-Schmidt 直交化



numactl あり（１ノード（ピュアMPI））



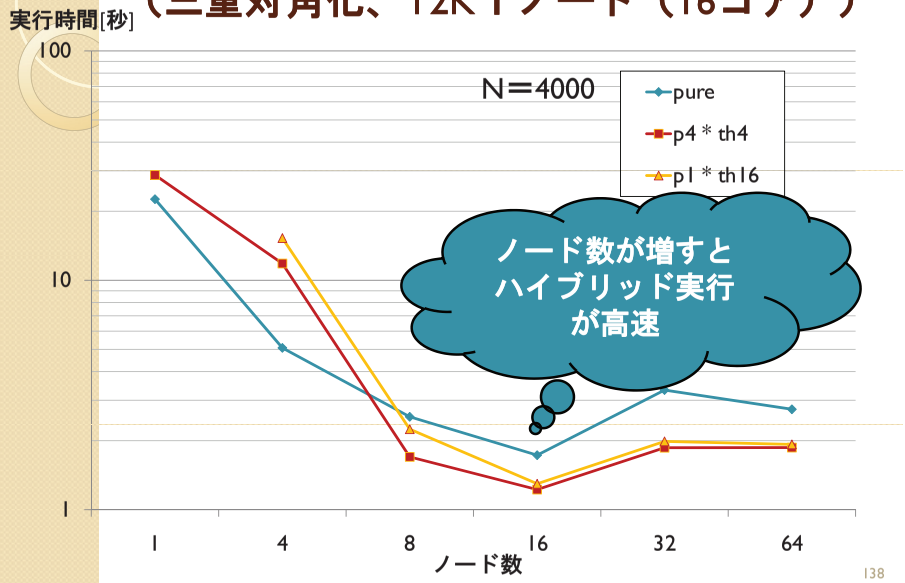
話の流れ

- 背景
- 対称実数密行列固有値ソルバ
  - アルゴリズムの説明
    - Householder三重対角化を経由するアルゴリズム
    - 先進的固有値アルゴリズム：MRRR
  - 性能評価
    - 性能評価環境：T2Kオープンスパコン（東大版）
    - 単体性能評価
    - ピュアMPI実行 vs ハイブリッド実行
    - 大規模問題実行性能

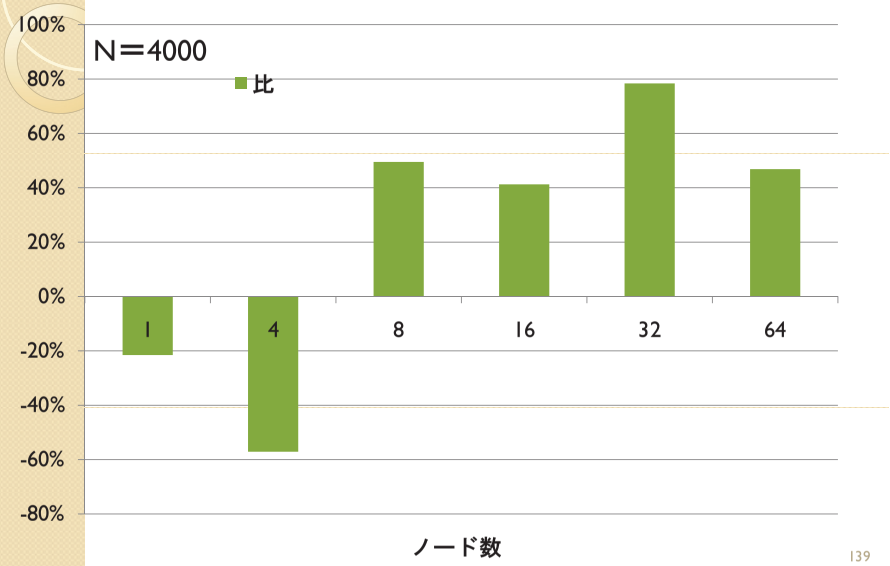
ピュアMPI 対 ハイブリッドMPI

- 計算機アーキテクチャの変化
- マルチコア環境（１チップ内（ソケット内）に複数の計算コアを積む環境）
  - ソケット内の通信速度は、ソケット外の通信速度より高速
- ソケット内はスレッド実行、ソケット外（ノード外）はMPI実行
  1. 実行形態が計算機アーキテクチャに合う
  2. MPIプロセス数減少により通信時間が削減
  3. ccNUMA環境で、ローカルメモリを有効に使える

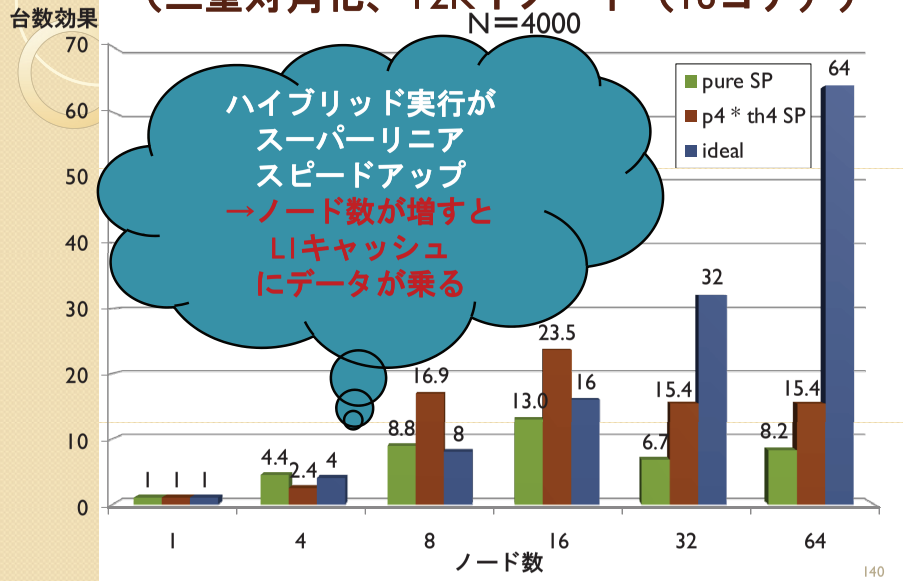
ハイブリッド実行の効果
（三重対角化、T2K１ノード（16コア））



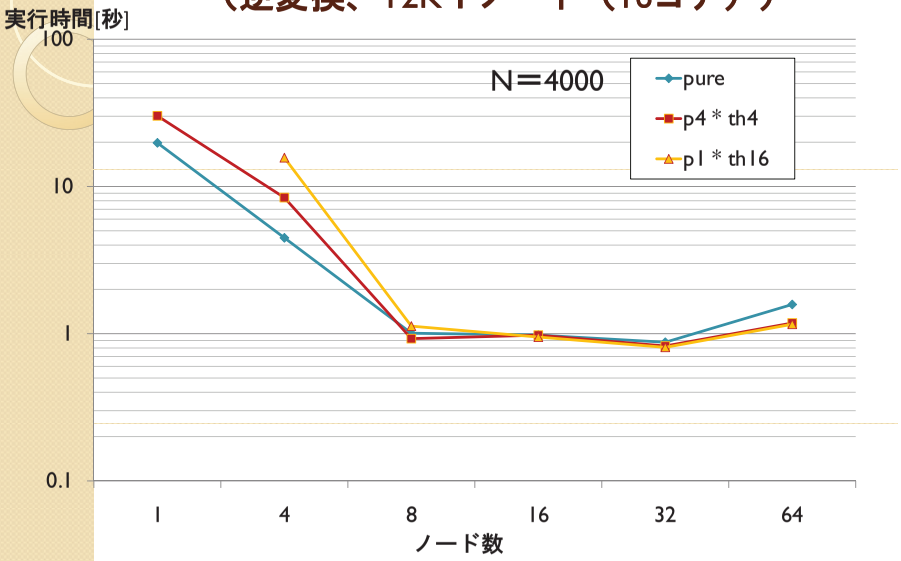
ハイブリッド実行の効果
（三重対角化、T2K１ノード（16コア））



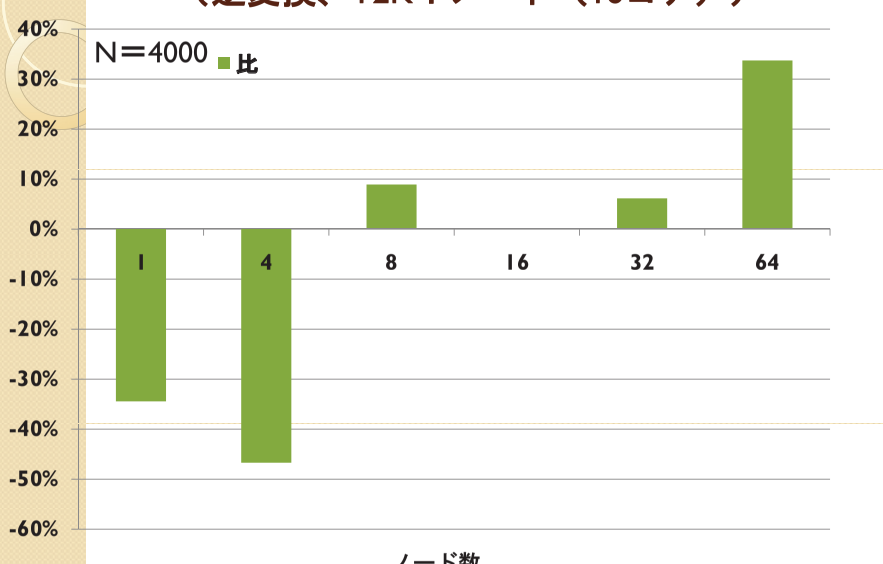
ハイブリッド実行の効果
（三重対角化、T2K１ノード（16コア））



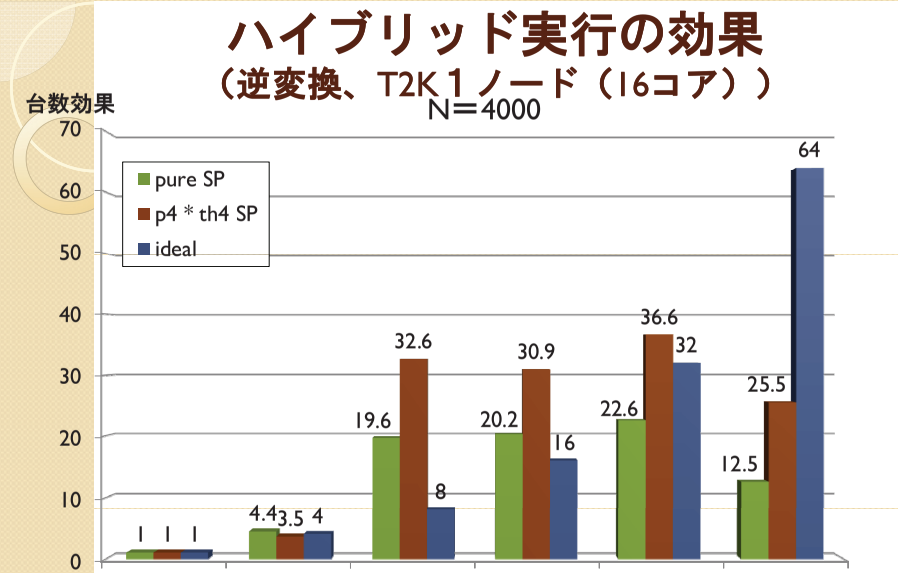
ハイブリッド実行の効果
（逆変換、T2K１ノード（16コア））



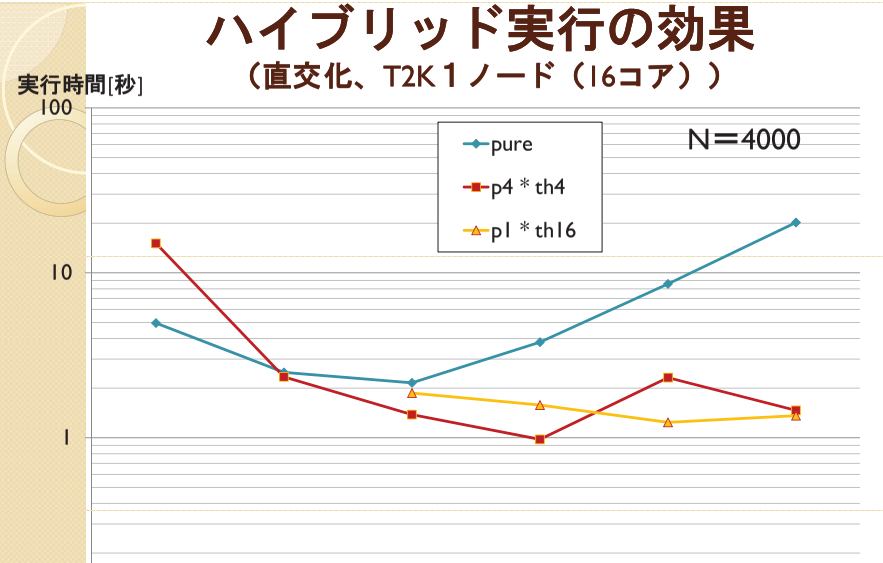
ハイブリッド実行の効果
（逆変換、T2K１ノード（16コア））



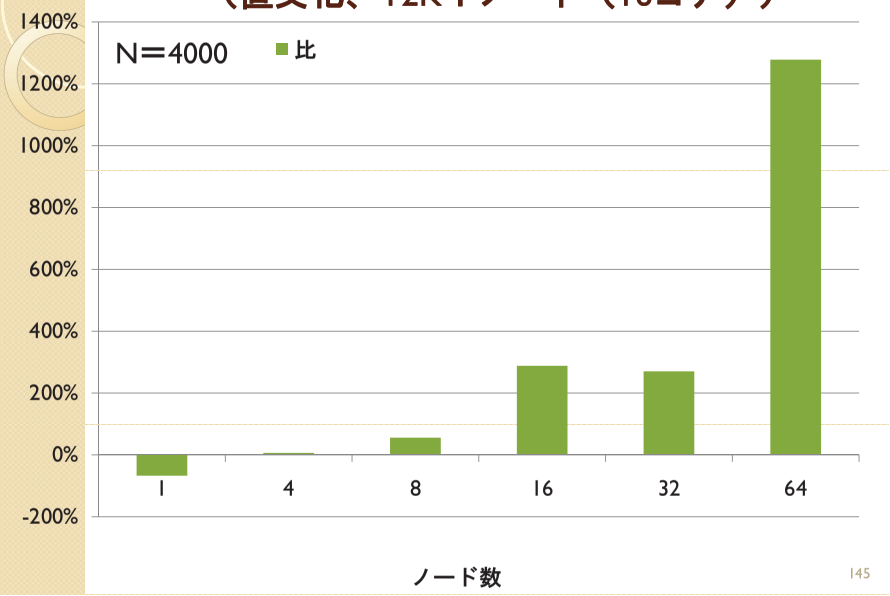
ハイブリッド実行の効果
（逆変換、T2K１ノード（16コア））



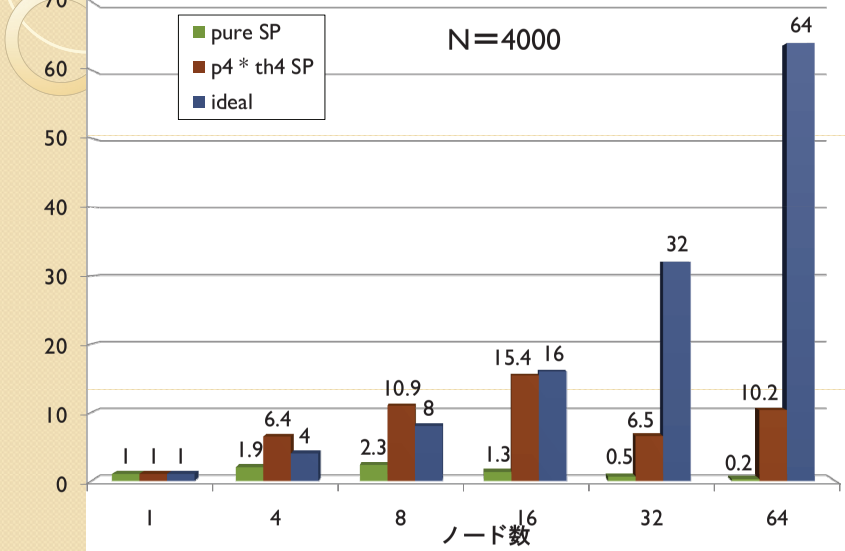
ハイブリッド実行の効果
（直交化、T2K１ノード（16コア））



ハイブリッド実行の効果
（直交化、T2K１ノード（16コア））



ハイブリッド実行の効果
（直交化、T2K１ノード（16コア））



話の流れ

- 背景
- 対称実数密行列固有値ソルバ
  - アルゴリズムの説明
    - Householder三重対角化を経由するアルゴリズム
    - 先進的固有値アルゴリズム：MRRR
  - 性能評価
    - 性能評価環境：T2Kオープンスパコン（東大版）
    - 単体性能評価
    - ピュアMPI実行 vs ハイブリッド実行
    - **大規模問題実行性能**

試験行列の説明

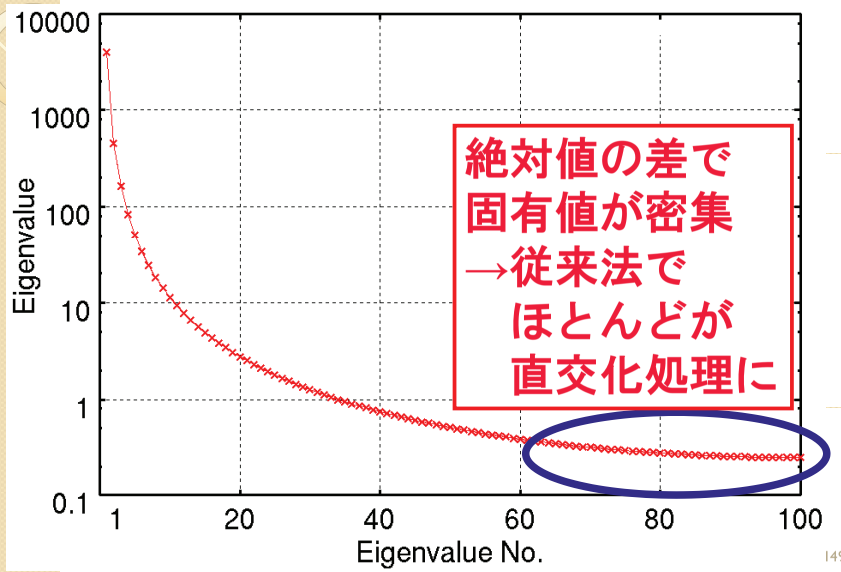
- Frank行列

$$A_n = \begin{bmatrix} n & n-1 & n-2 & \cdots & 1 \\ n-1 & n-1 & n-2 & \cdots & 1 \\ n-2 & n-2 & n-2 & \cdots & 1 \\ \vdots & \vdots & \vdots & \ddots & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

この行列の固有値は解析的に求められる

$$\lambda_k = \frac{1}{2(1-\cos\frac{2k-1}{2n+1}\pi)}, k = 1,2,\ldots, n$$

試験行列の固有値分布



固有値問題の精度検定

1. 固有値の解析解との差の最大値

$$\max_i \left| \hat{\lambda}_i - \lambda_i \right|, i = 1,2,\ldots, n$$

$\hat{\lambda}_i$:計算固有値　　λ_i : 解析固有値

2. 固有ベクトルの直交性

$$\left| X^T X - I \right|_F,$$

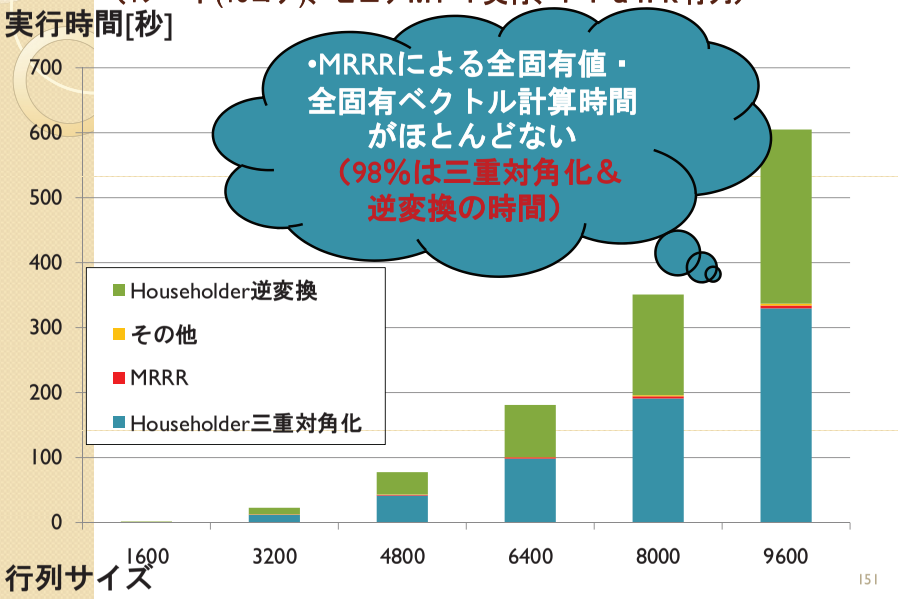
$$|\bullet|_F := \sqrt{\sum_{i,j} \hat{x}_i \hat{x}_j}$$

3. 元の行列の残差ベクトルの最大値

$$\max_i \left| A\hat{x}_i - \hat{\lambda}_i \hat{x}_i \right|_2, \ i = 1,2,\ldots, n$$

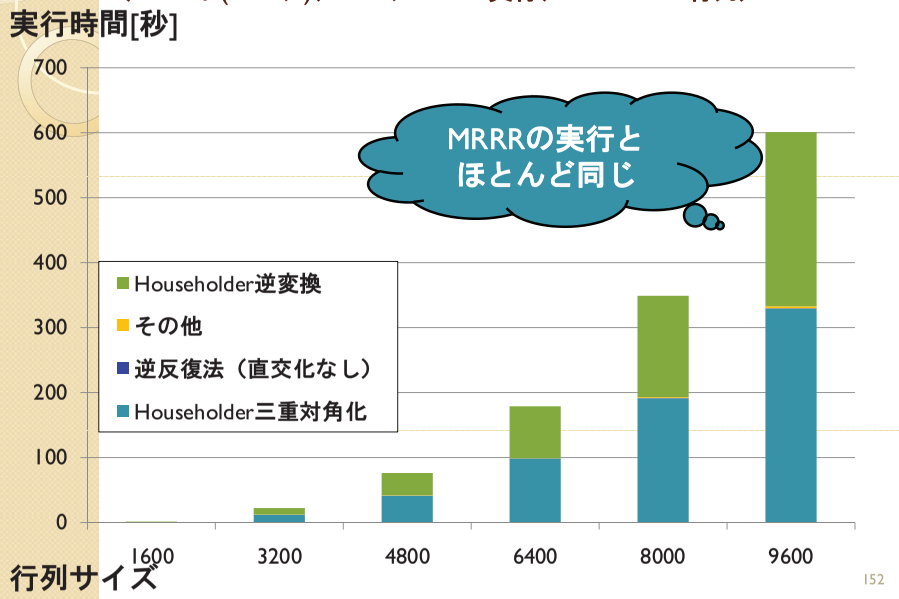
固有値ソルバ実行時間 [MRRR]

（1ノード(16コア)、ピュアMPI実行、Frank行列）



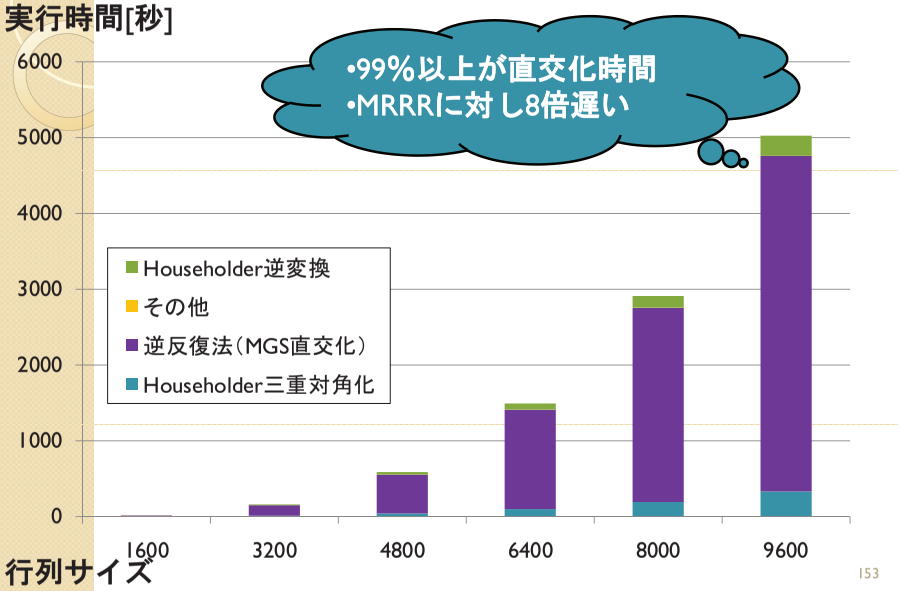
固有値ソルバ実行時間 [逆反復法（直交化なし）]

（1ノード(16コア)、ピュアMPI実行、Frank行列）



固有値ソルバ実行時間 [逆反復法（MGS直交化）]

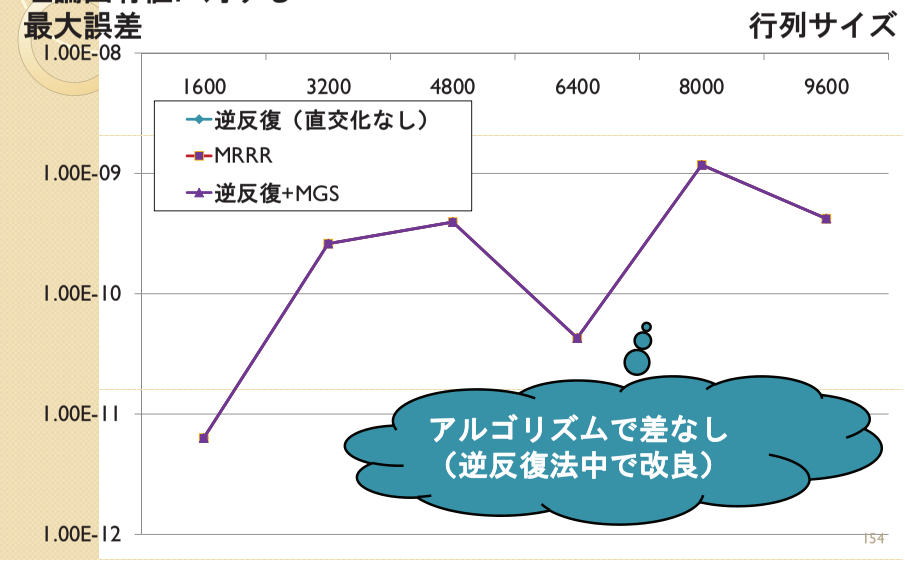
（1ノード(16コア)、ピュアMPI実行、Ｆｒａｎｋ行列）



153

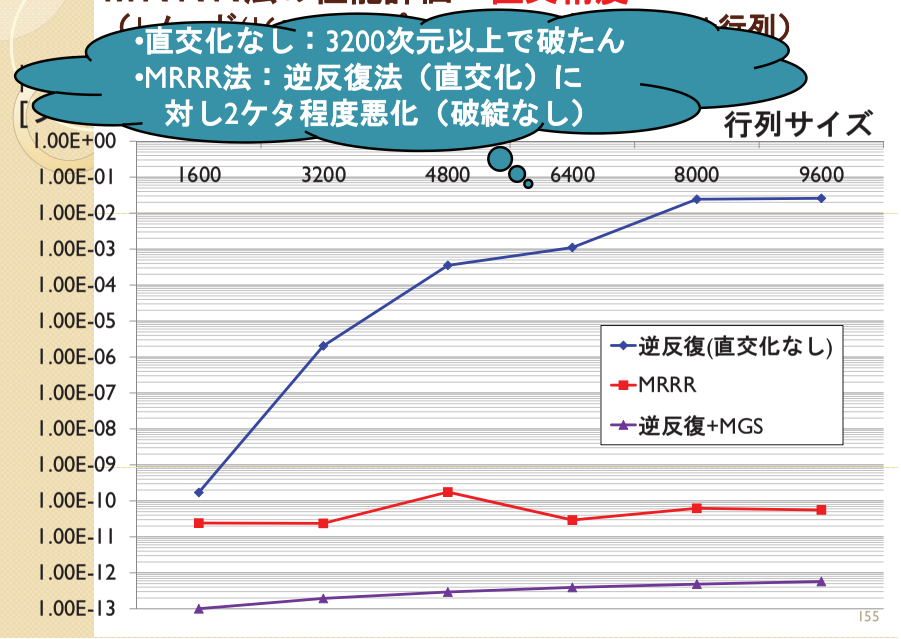
ＭＲＲＲ法の性能評価：固有値精度

（1ノード(16コア)、ピュアMPI実行、Frank行列）



154

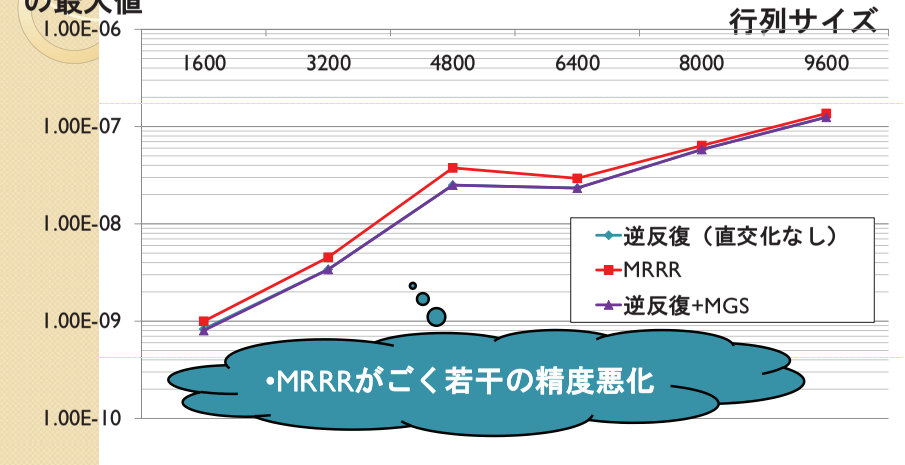
ＭＲＲＲ法の性能評価：直交精度



155

ＭＲＲＲ法の性能評価：残差ベクトル

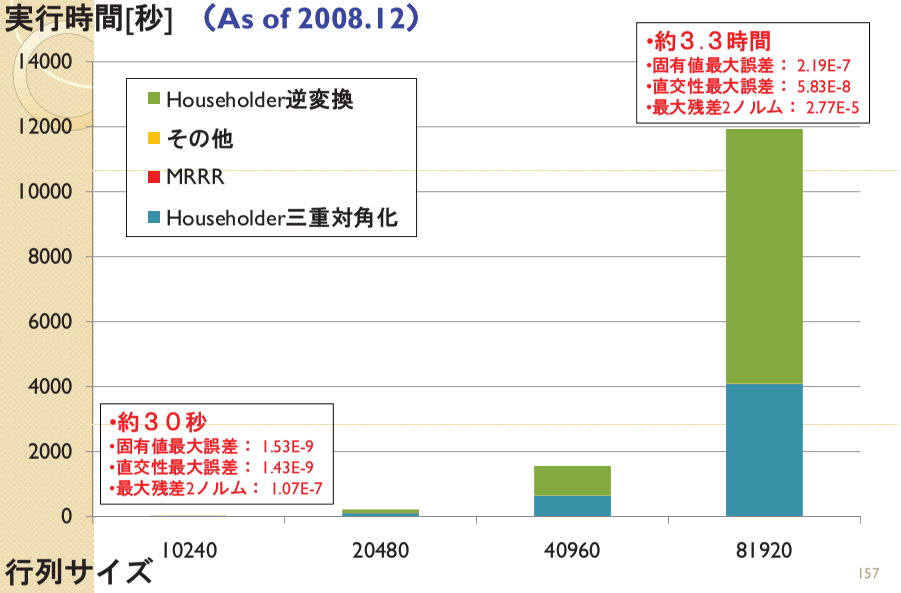
（1ノード(16コア)、ピュアMPI実行、Frank行列）



156

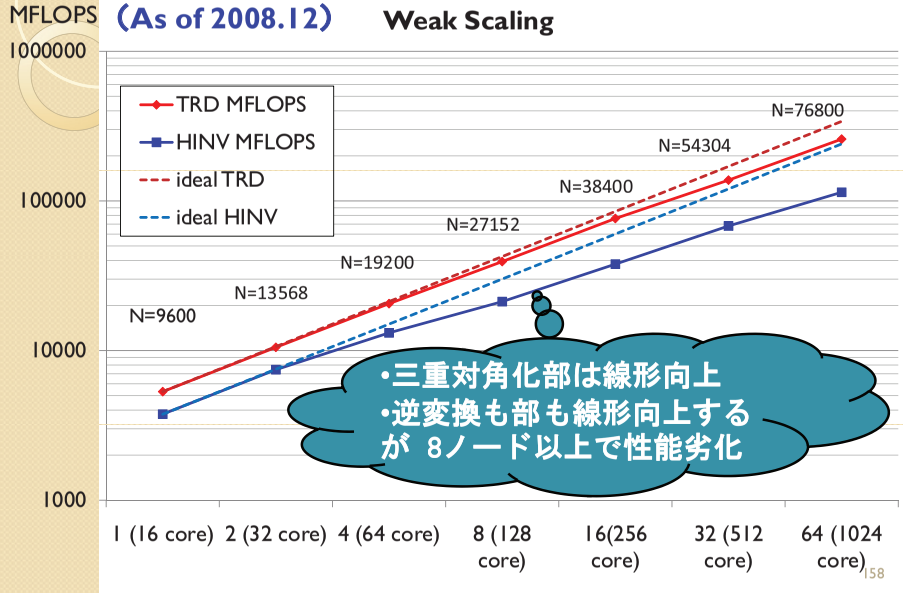
固有値ソルバ大規模問題実行時間 [MRRR]

（64ノード(1,024コア)、ピュアＭＰＩ実行、Ｆｒａｎｋ行列）



固有値ソルバ台数効果（weak scaling）

（1ノード当たりＮ＝9600、ピュアＭＰＩ実行、Ｆｒａｎｋ行列）



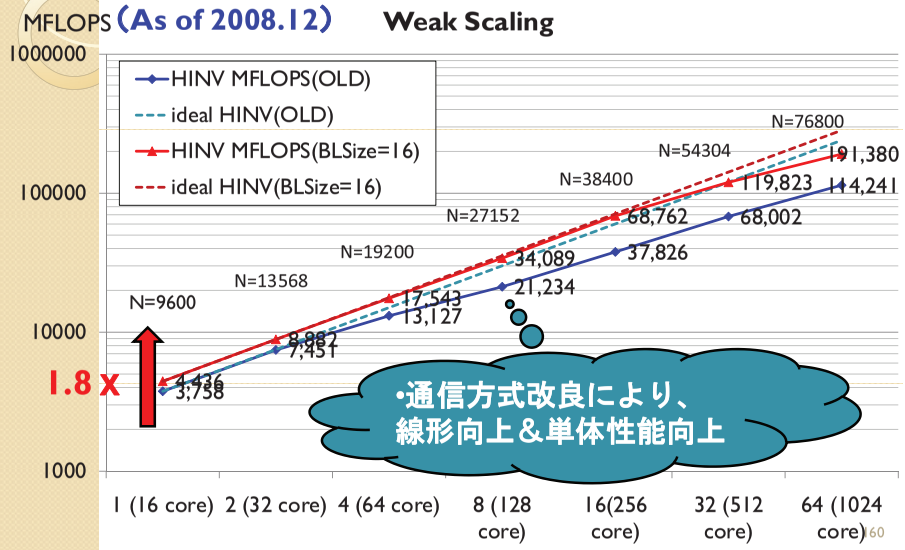
逆変換部分の通信方式改良

1. 通信ブロック化
   - 前提：Householder変換ベクトルが２次元サイクリック分散されている
   - 上記の収集をm個まとめて集める
   - まとめた分を同時に逆変換
   - 効果
     - 逆変換処理の時間が増し、通信時間が相対的に削減
       →より線形向上
     - 演算カーネルをアクセスが局所化するように変更
       効率の良い実装が可能
       →単体性能の向上
2. ベクトル長削減
   - 必要なベクトル長だけ集めるように改良
     →ベクトル長の削減による通信時間削減

固有値ソルバ台数効果（weak scaling）[MRRR]

（1ノード当たりＮ＝9600、ピュアＭＰＩ実行、Ｆｒａｎｋ行列）

（Householder逆変換：16個のベクトル毎に通信＆演算ブロック化版）



逆変換部分の演算性能改良法

- ループ交換によるキャッシュ最適化
  - 前提：通信ベクトル化が実装されており、カーネルが3重ループ化している

•変更前

```
do jbk=j, jend, -1
   do i=nstart, nend
      dot(norm, bufM(fb(jbk)), W(1, fi(i) )
      update(W(1,fi(i)), norm, bufM(fb(jbk)))
   enddo
enddo
```

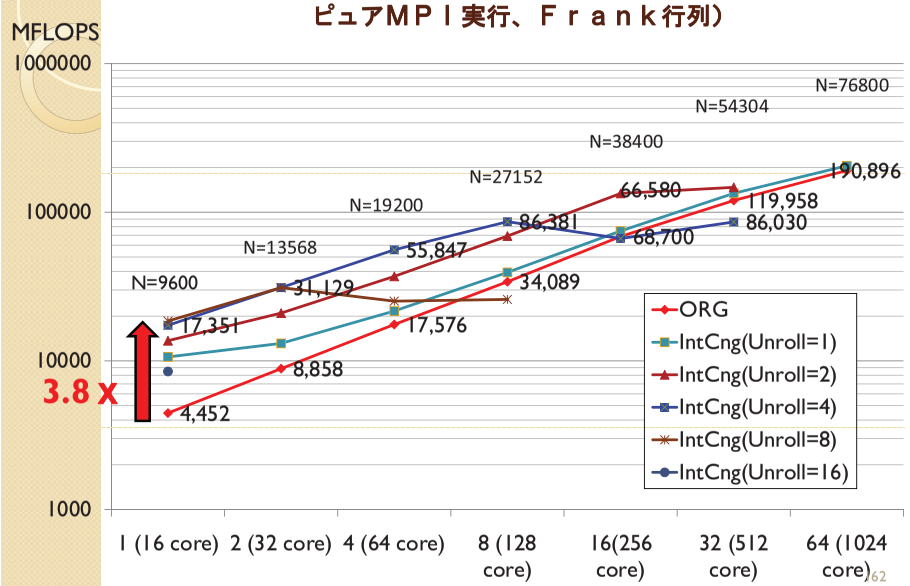
•変更後

```
do i=nstart, nend
  do jbk=j, jend, -1
     dot(norm, bufM(fb(jbk)), W(1, fi(i) )
     update(W(1,fi(i)), norm, bufM(fb(jbk)))
  enddo
enddo
```

ベクトル長（jbk~n), jbk=1,n-2がキャッシュに乗ると、再利用により速度向上可能

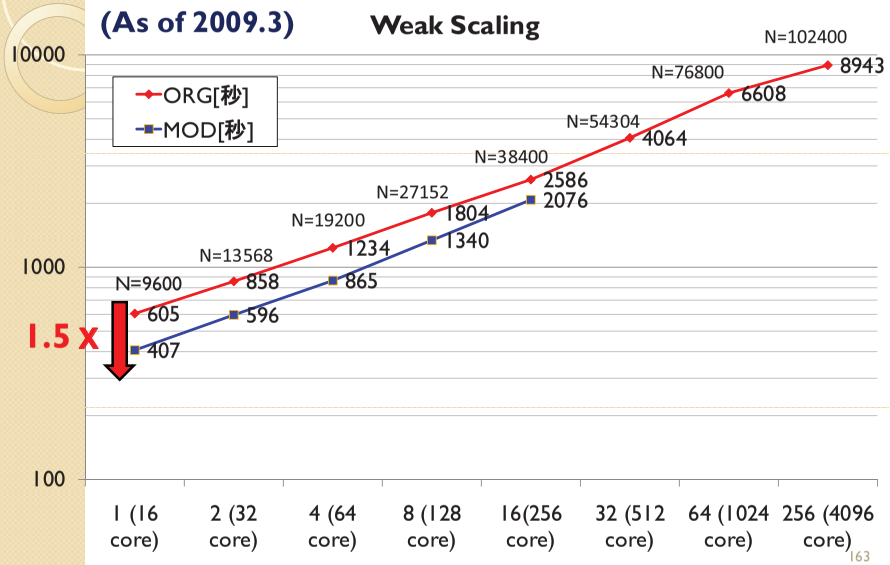
Householder逆変換実行性能（weak scaling）

（1ノード当たりN＝9600、通信ブロック幅＝16、ピュアMPI実行、Frank行列）



固有値ソルバ実行時間（weak scaling）[MRRR]

（1ノード当たりN＝9600、ピュアMPI実行、Frank行列）



固有値ソルバのまとめ

1. 先進的アルゴリズムMRRR法の導入
   - ２桁程度の固有ベクトル直交性悪化が許容可
     - 従来法に対し少なくとも8倍は高速化
     - 固有ベクトル計算部分について理論的に通信が不要、直交化なしの逆反復法と同等
       →10万並列に耐えられるアルゴリズム
   - 約10万次元の全固有値・全固有ベクトル計算時間
     - T2Kオープンスパコン256ノード（4096コア）で約２時間３０分
     - チューニングを全くしていない性能。
     - 2倍～3倍は高速化できる。→１時間以内に
2. ハイブリッドMPI実行
   - マルチコア環境での技術の適用により、4割程度高速化できる例

残された話題

- マルチコア向きの新アルゴリズム
  1. TSQR(Tall Skinny QR) [A.Buttari et.al. 2007]
     - 部分行列のQR分解が独立に行える
     - 全体のQR分解が、２分木形態の通信で完成
       （小行列の行列 - 行列積のみで構築可能）

       →高い並列性とマルチコアのキャッシュ階層に合う演算パターン

$$A=\begin{pmatrix}A_0\\A_1\\A_2\\A_3\end{pmatrix}=\begin{pmatrix}Q_{00}R_{00}\\Q_{10}R_{10}\\Q_{20}R_{20}\\Q_{30}R_{30}\end{pmatrix}=\begin{pmatrix}Q_{00}&&&\\&Q_{10}&&\\&&Q_{20}&\\&&&Q_{30}\end{pmatrix}\begin{pmatrix}R_{00}\\R_{10}\\R_{20}\\R_{30}\end{pmatrix}=\begin{pmatrix}Q_{01}R_{01}\\Q_{11}R_{11}\end{pmatrix}=\dots$$

残された話題

- マルチコア向きの新アルゴリズム
  2. 帯行列を経由するHouseholder変換による固有値ソルバ[今村ら、2008]
     - Householder三重対角化を帯行列で止める
     - 帯行列の全固有値・全固有ベクトルを分割統治法で求解

       →三重対角化部の通信量削減による並列化効率向上
     - ブロック化によりＴ２Ｋでピーク性能比30%超を達成
     - 事前評価として、三重対角化におけるブロック化版を実装
       - 8万次元：64ノード実行で20分弱で終わる

今村俊幸：マルチコア環境における固有値ソルバ、計算工学講演会論文集、Vol.14、pp.171-174 (2009年5月)